Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	ChiliZ (CHZ) Token
Website:	chiliz.com
Platform:	Ethereum
Language:	Solidity
Date:	February 19th, 2024

Table of contents

Introduction 4
Project Background 4
Audit Scope 4
Claimed Smart Contract Features 5
Audit Summary 6
Technical Quick Stats 7
Business Risk Analysis 8
Code Quality 9
Documentation 9
Use of Dependencies 9
AS-IS overview 10
Severity Definitions 12
Audit Findings 13
Conclusion 19
Our Methodology 20
Disclaimers 22
Appendix
- Code Flow Diagram 23
- Slither Results Log 24
- Solidity static analysis 25
- Solhint Linter 26

`

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 ChiliZ Token smart contract from chiliz.com 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 February 19th, 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 ChiliZ is an ERC20-based smart contract that has functions that allows children to implement an emergency stop mechanism.
- The provided Solidity code defines a comprehensive ERC20 token contract called `chiliZ (CHZ)`. This contract includes several features such as pausable token transfers, safe math operations, and role-based permissions.
- This comprehensive implementation ensures the `chiliZ` token is robust, and secure, and adheres to best practices in smart contract development.

Audit scope

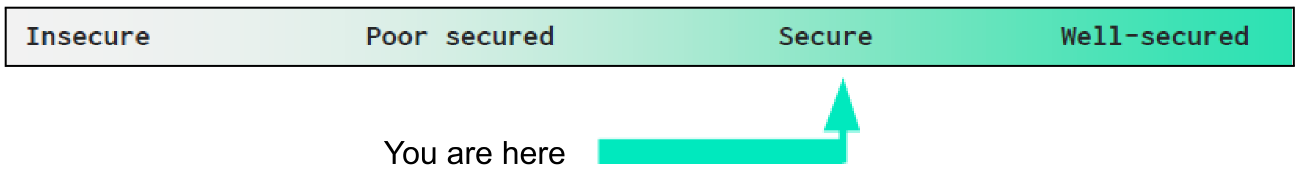
Name	**Code Review and Security Analysis Report for ChiliZ Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	chiliZ.sol
Ethereum Code	0x3506424f91fd33084466f402d5d97f05f8e3b4af
Audit Date	February 19th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: ChiliZ • Symbol: CHZ • Decimals: 18 • Total Supply:1 billion	**YES, This is valid.**
Ownership Control: • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**
Other Specifications: • Pause / Unpause contract. • Pause / Unpause contract state.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured".** This token contract does not have any ownership control, hence it is **100% 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	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack 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	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	Not Detected
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● 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 contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in ChiliZ 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 ChiliZ 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 ChiliZ 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 are 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	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	transfer	write	Missing required error message	Refer Audit Findings
6	transferFrom	write	Missing required error message	Refer Audit Findings
7	approve	write	Missing required error message	Refer Audit Findings
8	increaseAllowance	write	Missing required error message	Refer Audit Findings
9	decreaseAllowance	write	Missing required error message	Refer Audit Findings
10	paused	read	Passed	No Issue
11	whenNotPaused	modifier	Missing required error message	Refer Audit Findings
12	whenPaused	modifier	Missing required error message	Refer Audit Findings
13	pause	write	access only Pauser	No Issue
14	unpause	write	access only Pauser	No Issue
15	onlyPauser	modifier	Missing required error message	Refer Audit Findings
16	isPauser	read	Passed	No Issue
17	addPauser	write	access only Pauser	No Issue
18	renouncePauser	write	Passed	No Issue
19	_addPauser	internal	Passed	No Issue
20	_removePauser	internal	Passed	No Issue
21	totalSupply	read	Passed	No Issue
22	balanceOf	read	Passed	No Issue
23	allowance	read	Passed	No Issue
24	transfer	write	Passed	No Issue
25	approve	write	Passed	No Issue
26	transferFrom	write	Passed	No Issue
27	increaseAllowance	write	Passed	No Issue
28	decreaseAllowance	write	Passed	No Issue
29	_transfer	internal	Missing required error message	Refer Audit Findings
30	_mint	internal	Missing required error message	Refer Audit Findings
31	_burn	internal	Missing required error message	Refer Audit Findings

32	_burnFrom	internal	Missing required error message	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:

```
pragma solidity ^0.4.24;
```

Using the latest solidity will prevent any compiler-level bugs.

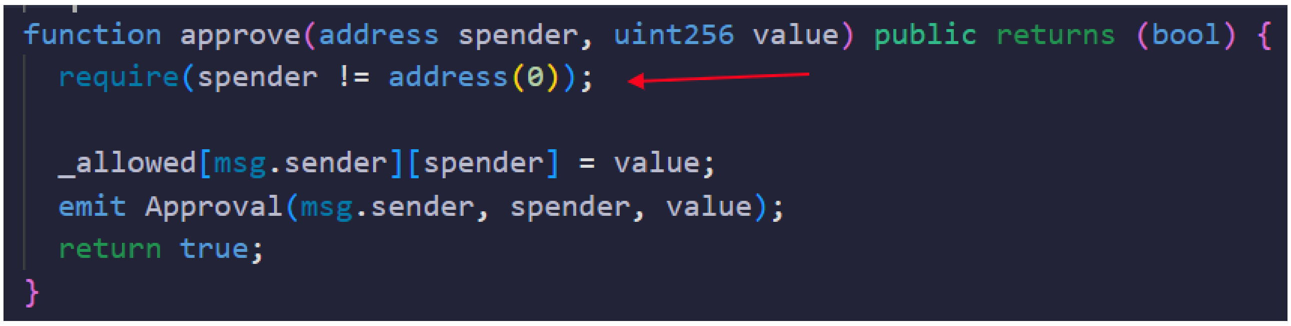
Resolution: We suggest using the latest solidity compiler version.

(2) Missing SPDX license identifier:

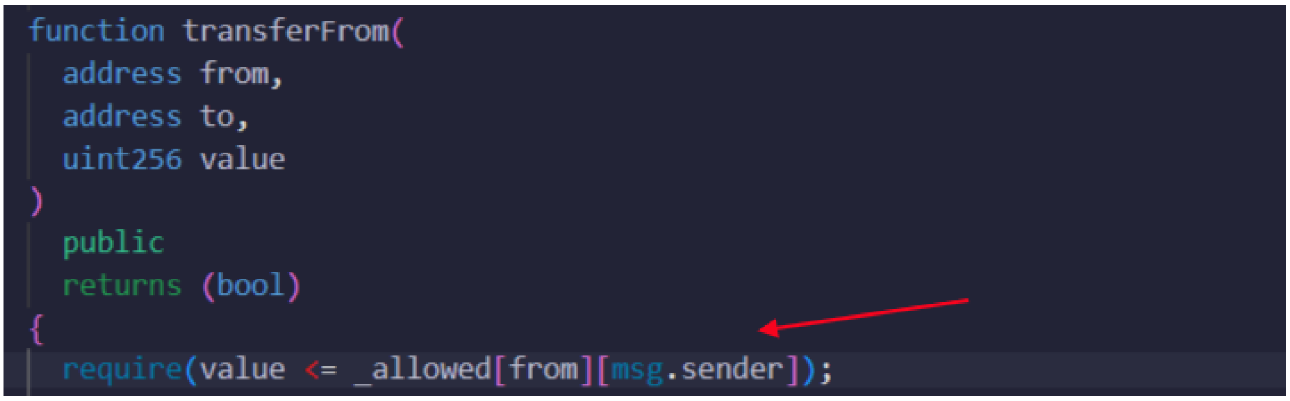
Solidity's new specification requires a valid SPDX license identifier to be included in every smart contract file.

Resolution: Please add a comment for the appropriate SPDX license identifier.

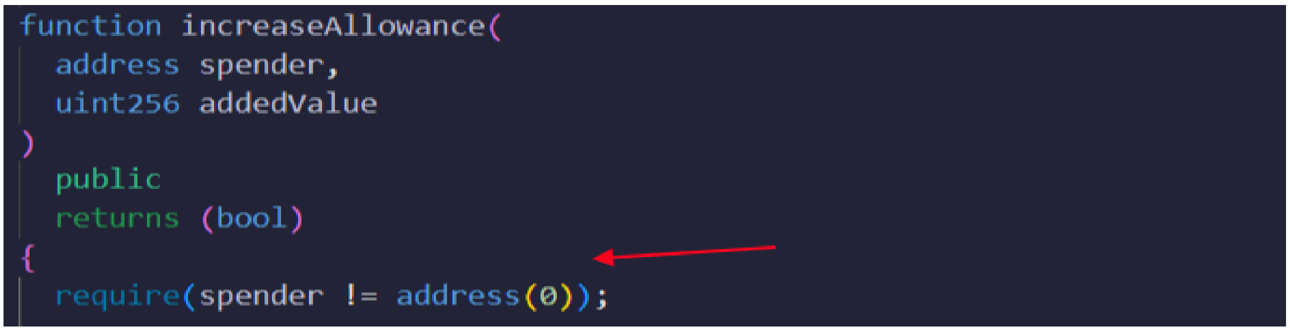
(3) Missing required error message:



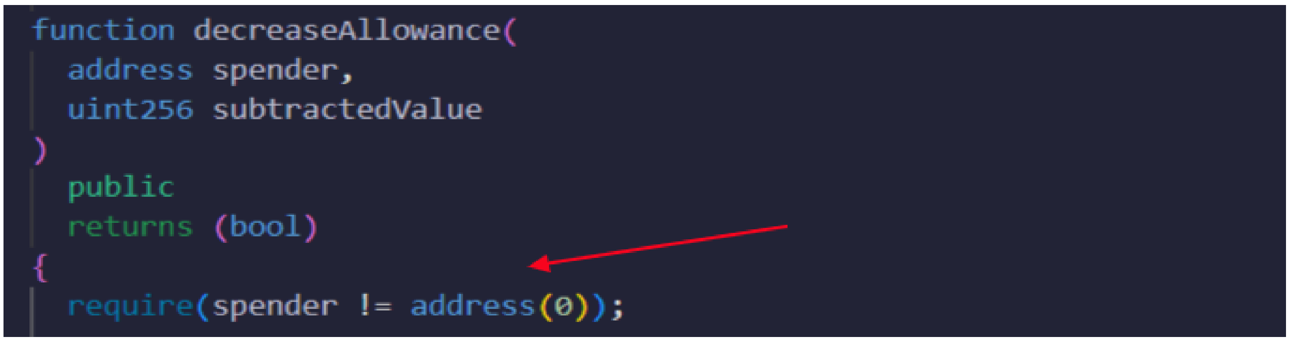
```
function approve(address spender, uint256 value) public returns (bool) {
  require(spender != address(0));

  _allowed[msg.sender][spender] = value;
  emit Approval(msg.sender, spender, value);
  return true;
}
```



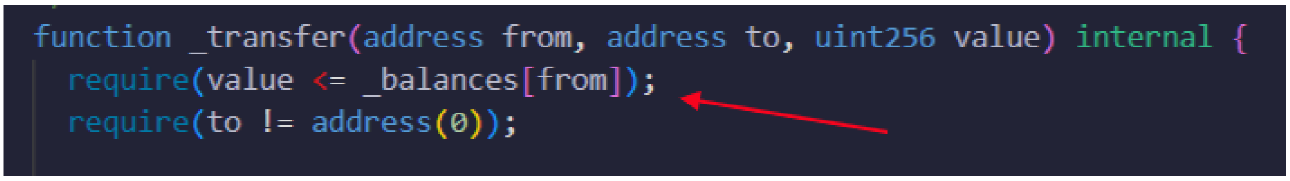
```
function transferFrom(
  address from,
  address to,
  uint256 value
)
  public
  returns (bool)
{
  require(value <= _allowed[from][msg.sender]);
```



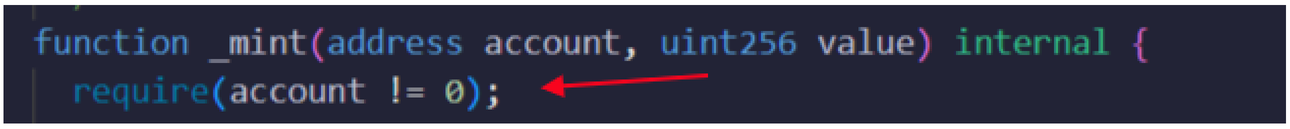
```
function increaseAllowance(
  address spender,
  uint256 addedValue
)
  public
  returns (bool)
{
  require(spender != address(0));
```



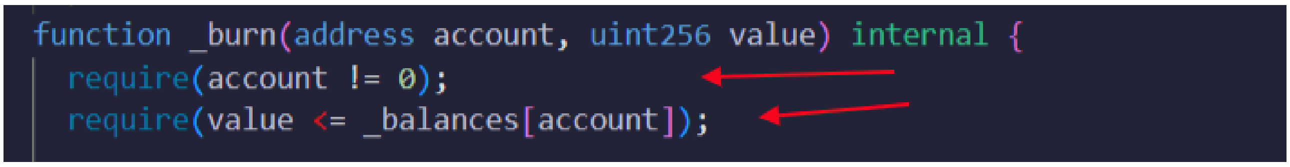
```
function decreaseAllowance(
  address spender,
  uint256 subtractedValue
)
  public
  returns (bool)
{
  require(spender != address(0));
```



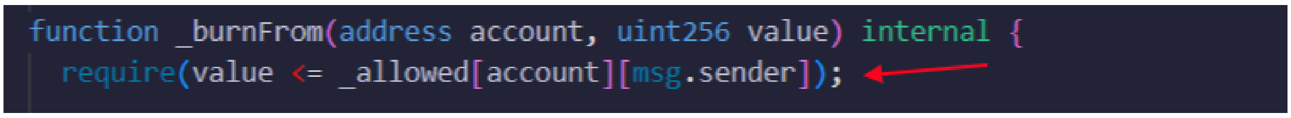
```
function _transfer(address from, address to, uint256 value) internal {
  require(value <= _balances[from]);
  require(to != address(0));
```



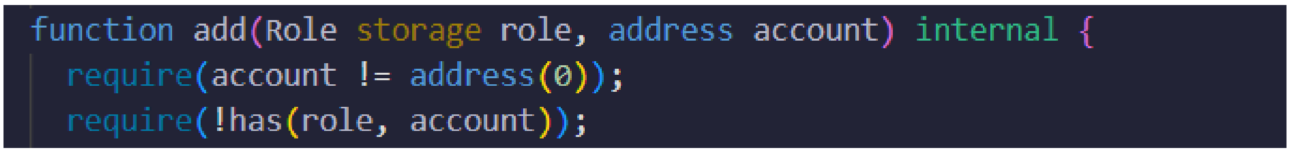
```
function _mint(address account, uint256 value) internal {
  require(account != 0);
```



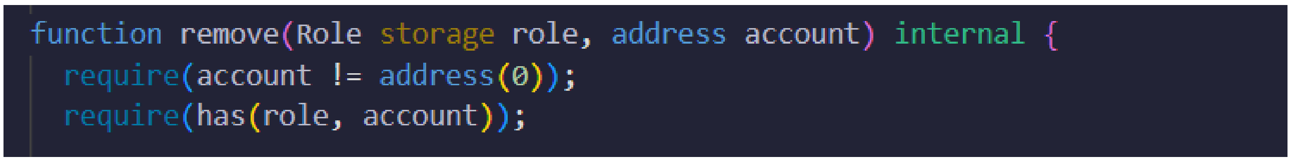
```
function _burn(address account, uint256 value) internal {
  require(account != 0);
  require(value <= _balances[account]);
```



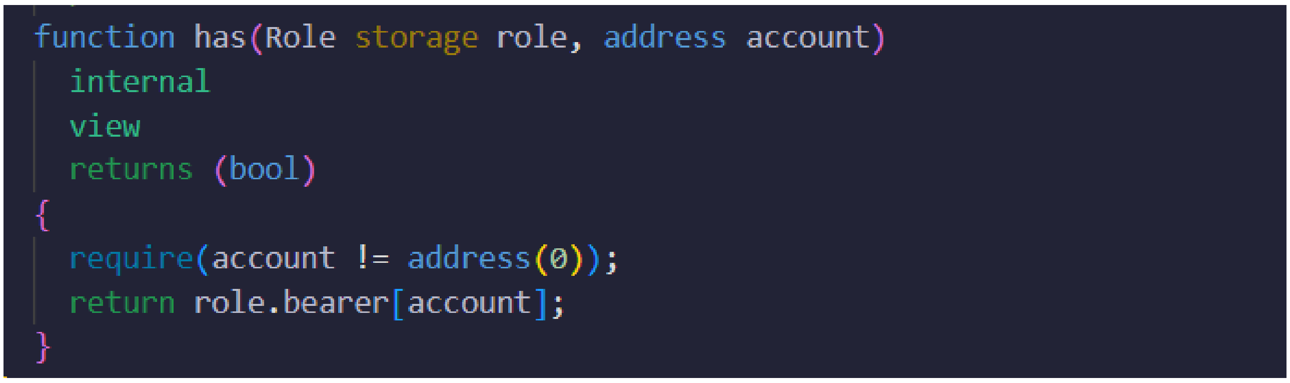
```
function _burnFrom(address account, uint256 value) internal {
  require(value <= _allowed[account][msg.sender]);
```



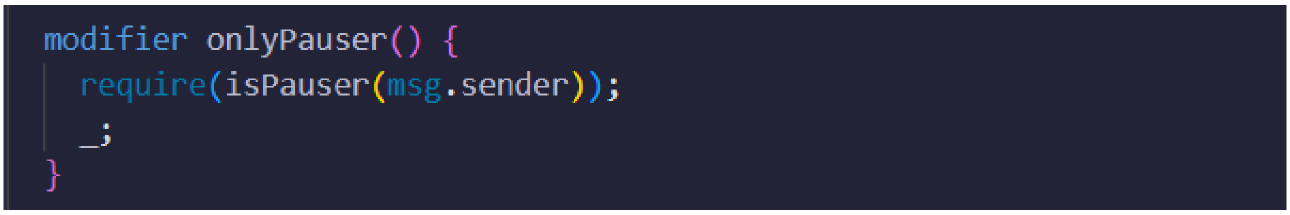
```
function add(Role storage role, address account) internal {
  require(account != address(0));
  require(!has(role, account));
```



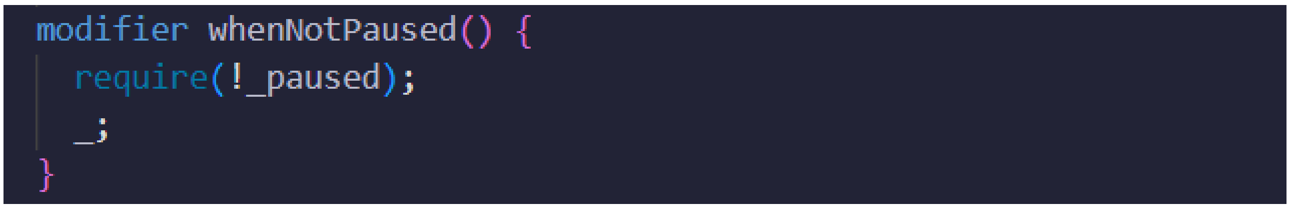
```
function remove(Role storage role, address account) internal {
  require(account != address(0));
  require(has(role, account));
```



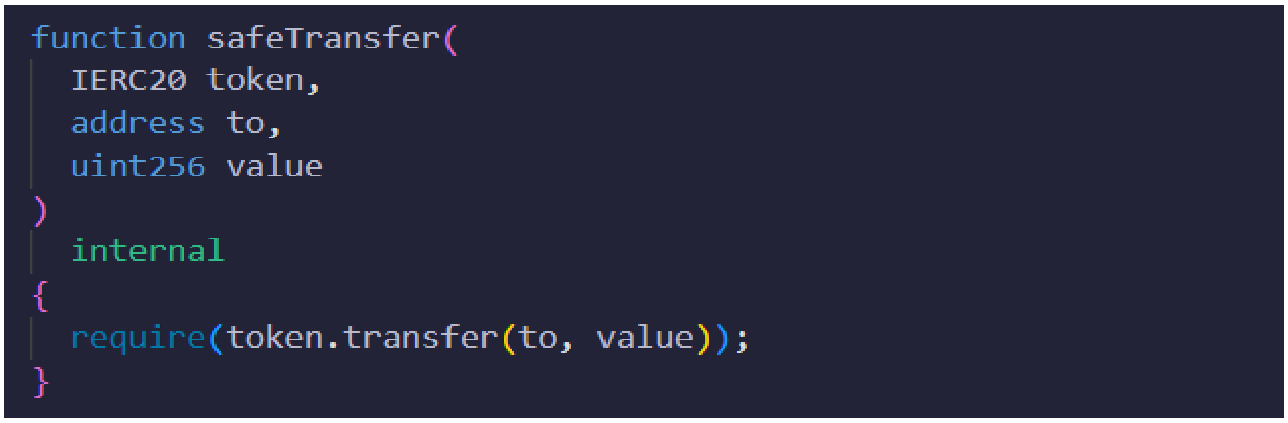
```
function has(Role storage role, address account)
  internal
  view
  returns (bool)
{
  require(account != address(0));
  return role.bearer[account];
}
```



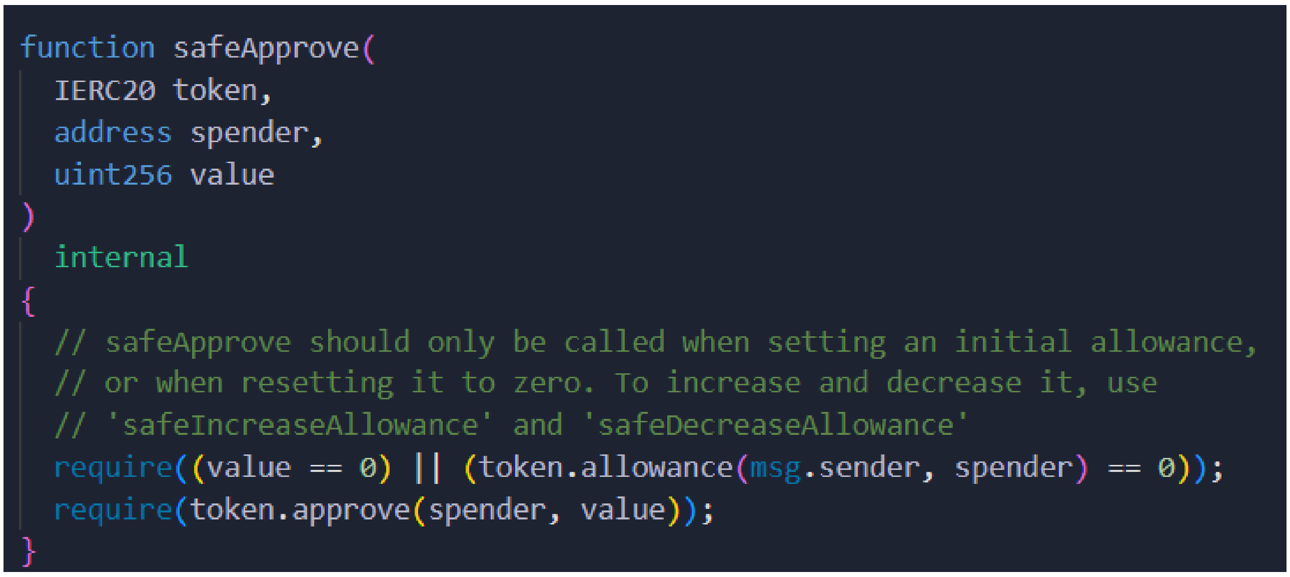
```
modifier onlyPauser() {
  require(isPauser(msg.sender));
  _;
}
```



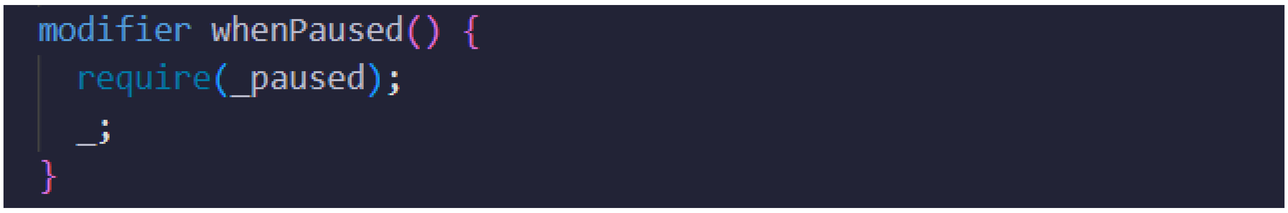
```
modifier whenNotPaused() {
  require(!_paused);
  _;
}
```



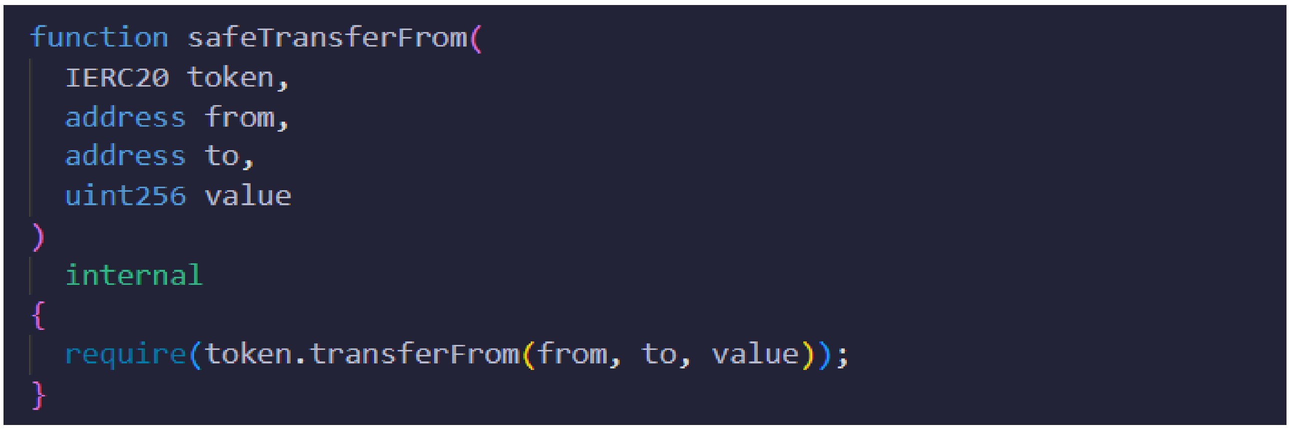
```
function safeTransfer(
  IERC20 token,
  address to,
  uint256 value
)
  internal
{
  require(token.transfer(to, value));
}
```



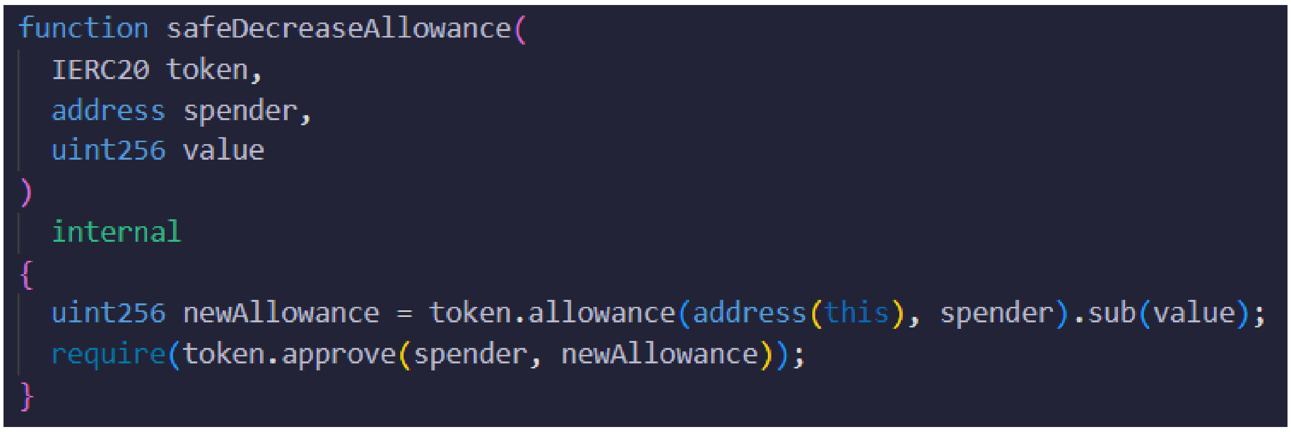
```
function safeApprove(
  IERC20 token,
  address spender,
  uint256 value
)
  internal
{
  // safeApprove should only be called when setting an initial allowance,
  // or when resetting it to zero. To increase and decrease it, use
  // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
  require((value == 0) || (token.allowance(msg.sender, spender) == 0));
  require(token.approve(spender, value));
}
```



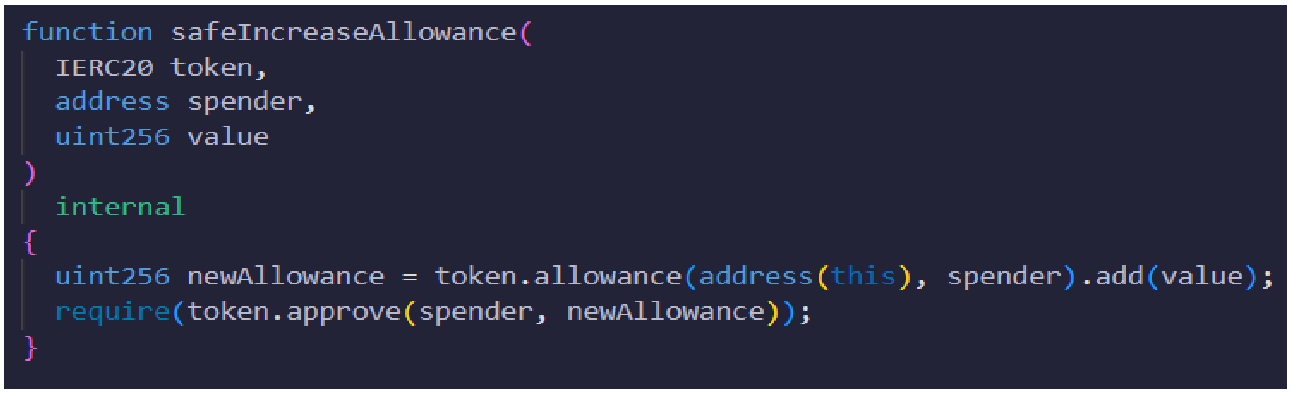
```
modifier whenPaused() {
  require(_paused);
  _;
}
```



```
function safeTransferFrom(
  IERC20 token,
  address from,
  address to,
  uint256 value
)
  internal
{
  require(token.transferFrom(from, to, value));
}
```



```
function safeDecreaseAllowance(
  IERC20 token,
  address spender,
  uint256 value
)
  internal
{
  uint256 newAllowance = token.allowance(address(this), spender).sub(value);
  require(token.approve(spender, newAllowance));
}
```

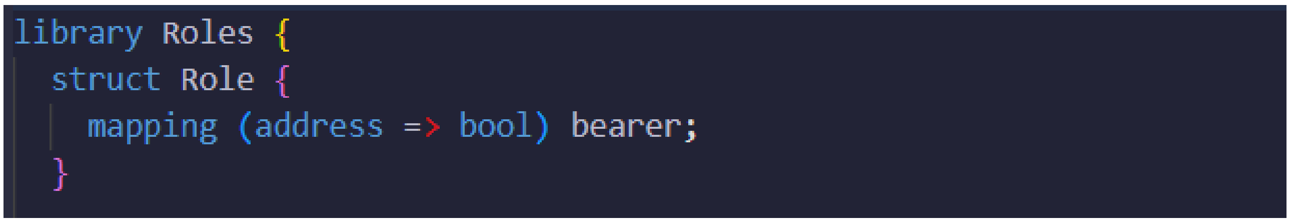


```
function safeIncreaseAllowance(
    IERC20 token,
    address spender,
    uint256 value
)
    internal
{
    uint256 newAllowance = token.allowance(address(this), spender).add(value);
    require(token.approve(spender, newAllowance));
}
```

Detecting missing errors in the whole require statement.

Resolution: We suggest providing a custom error message that will be displayed if the condition is not met. This can be helpful for debugging and understanding why a transaction failed.

(4) Explicit Visibility for State Variables Warning:



```
library Roles {
  struct Role {
    mapping (address => bool) bearer;
  }
```

The warning is related to the visibility of state variables in your Solidity code.

Resolution: We recommend updating the code to explicitly mark the visibility of state variables using the internal or public keyword, depending on the intended visibility.

Centralization Risk

The ChiliZ Token smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

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. but those 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).

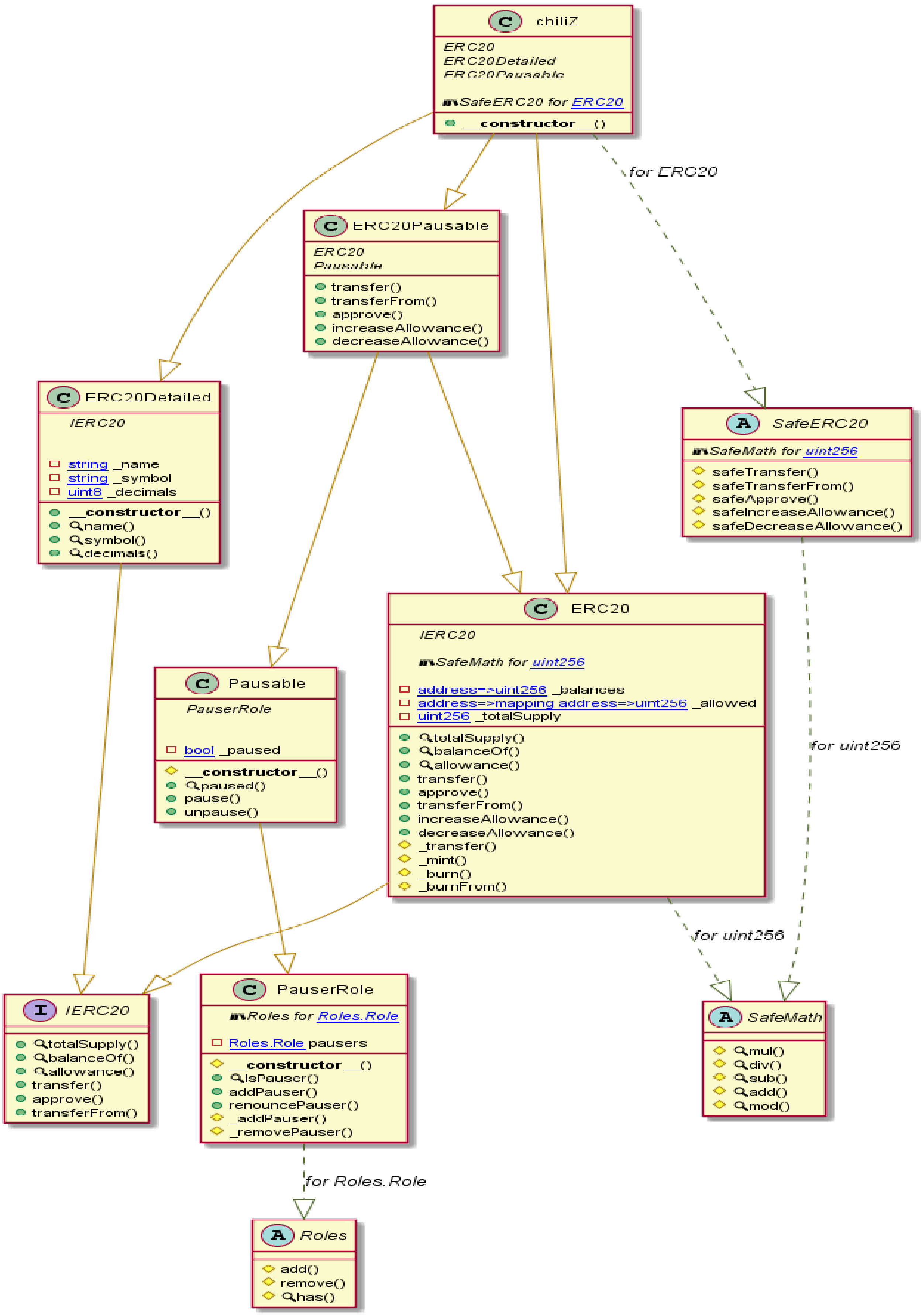
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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

Appendix

Code Flow Diagram - ChiliZ Token

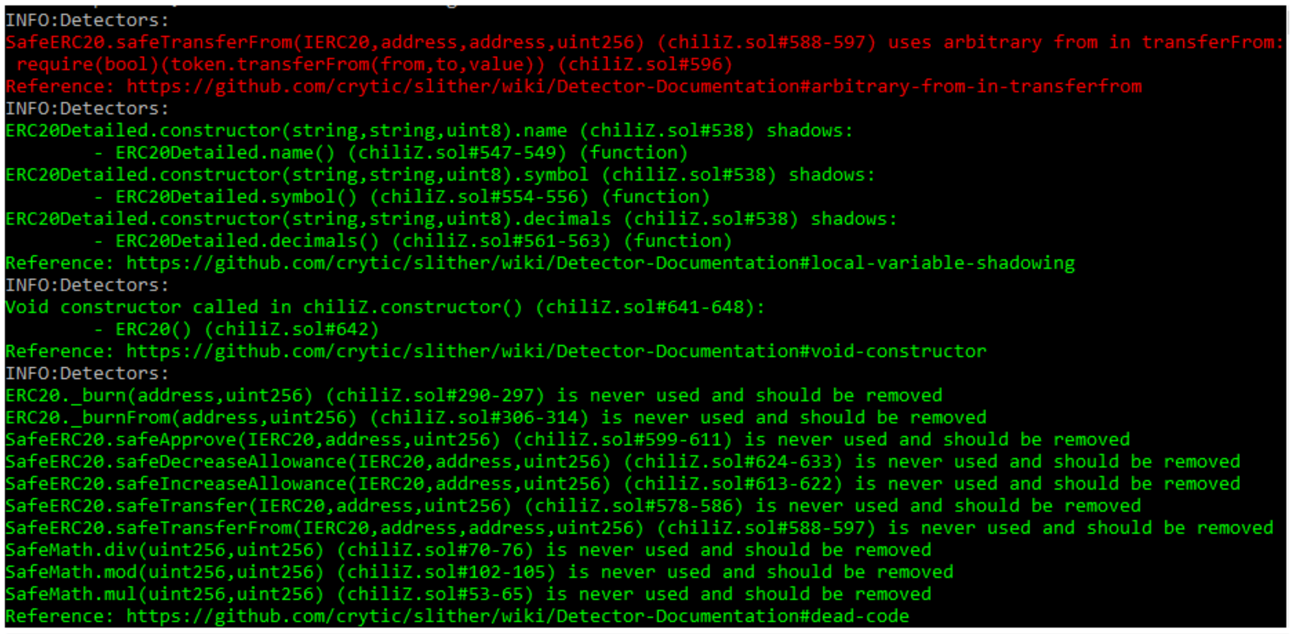


Slither Results Log

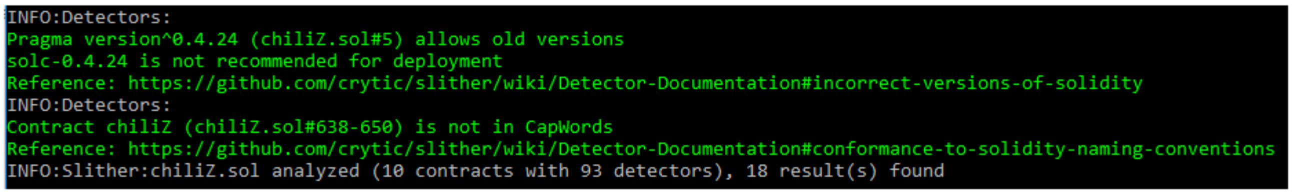
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report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



```
INFO:Detectors:
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (chiliZ.sol#588-597) uses arbitrary from in transferFrom:
 require(bool)(token.transferFrom(from,to,value)) (chiliZ.sol#596)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (chiliZ.sol#538) shadows:
        - ERC20Detailed.name() (chiliZ.sol#547-549) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (chiliZ.sol#538) shadows:
        - ERC20Detailed.symbol() (chiliZ.sol#554-556) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (chiliZ.sol#538) shadows:
        - ERC20Detailed.decimals() (chiliZ.sol#561-563) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Void constructor called in chiliZ.constructor() (chiliZ.sol#641-648):
        - ERC20() (chiliZ.sol#642)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#void-constructor
INFO:Detectors:
ERC20._burn(address,uint256) (chiliZ.sol#290-297) is never used and should be removed
ERC20._burnFrom(address,uint256) (chiliZ.sol#306-314) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (chiliZ.sol#599-611) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (chiliZ.sol#624-633) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (chiliZ.sol#613-622) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (chiliZ.sol#578-586) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (chiliZ.sol#588-597) is never used and should be removed
SafeMath.div(uint256,uint256) (chiliZ.sol#70-76) is never used and should be removed
SafeMath.mod(uint256,uint256) (chiliZ.sol#102-105) is never used and should be removed
SafeMath.mul(uint256,uint256) (chiliZ.sol#53-65) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

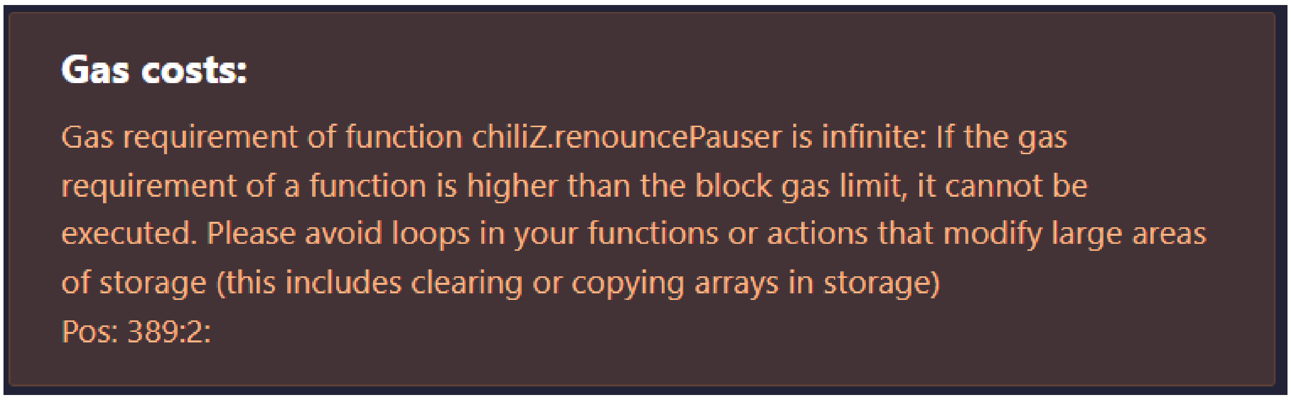


```
INFO:Detectors:
Pragma version^0.4.24 (chiliZ.sol#5) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Contract chiliZ (chiliZ.sol#638-650) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:chiliZ.sol analyzed (10 contracts with 93 detectors), 18 result(s) found
```

Solidity Static Analysis

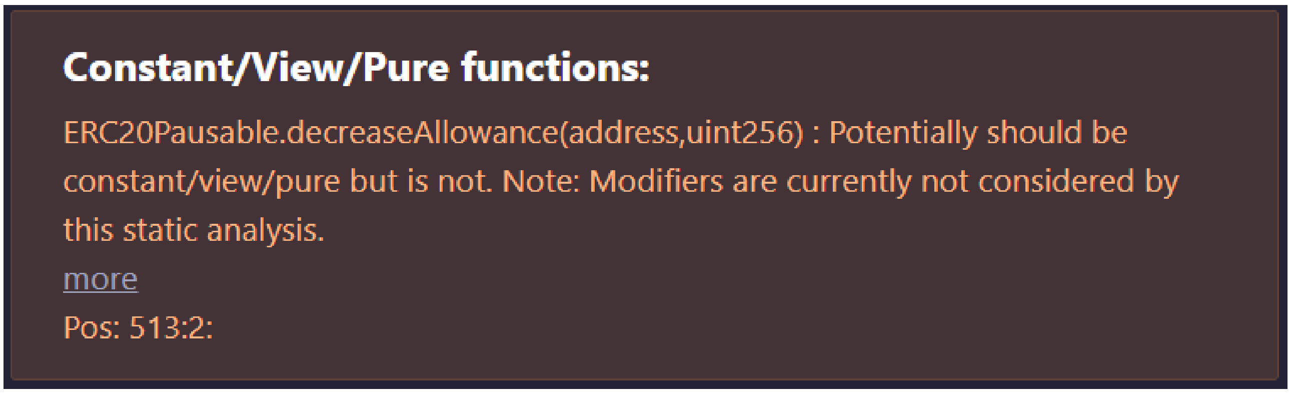
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

chiliZ.sol



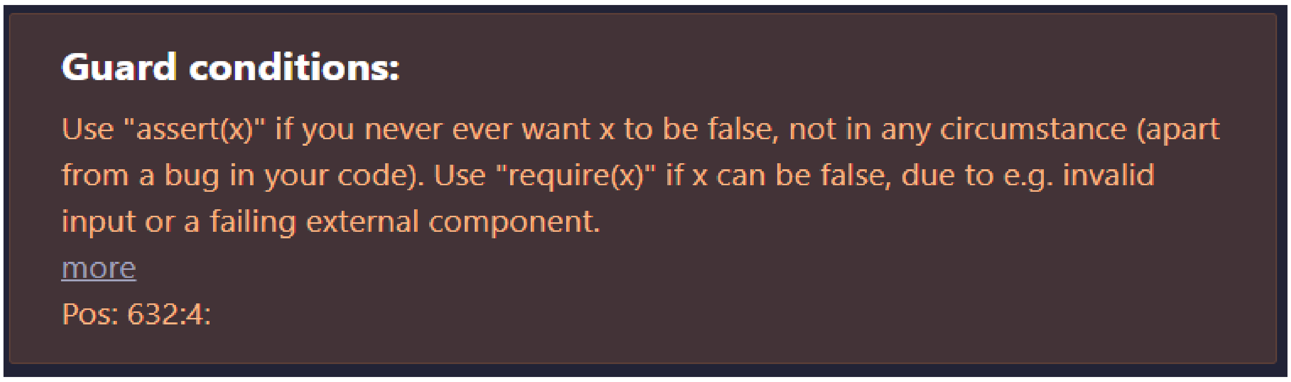
Gas costs:

Gas requirement of function chiliZ.renouncePauser 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: 389:2:



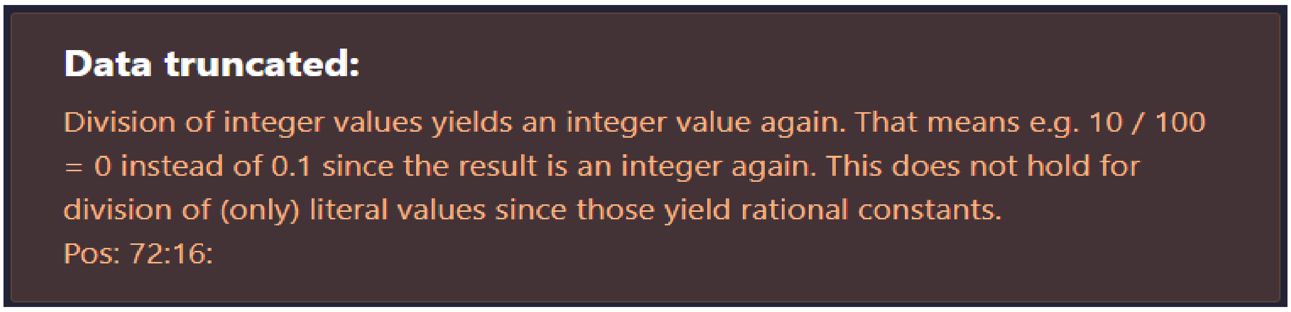
Constant/View/Pure functions:

ERC20Pausable.decreaseAllowance(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 513:2:



Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 632:4:



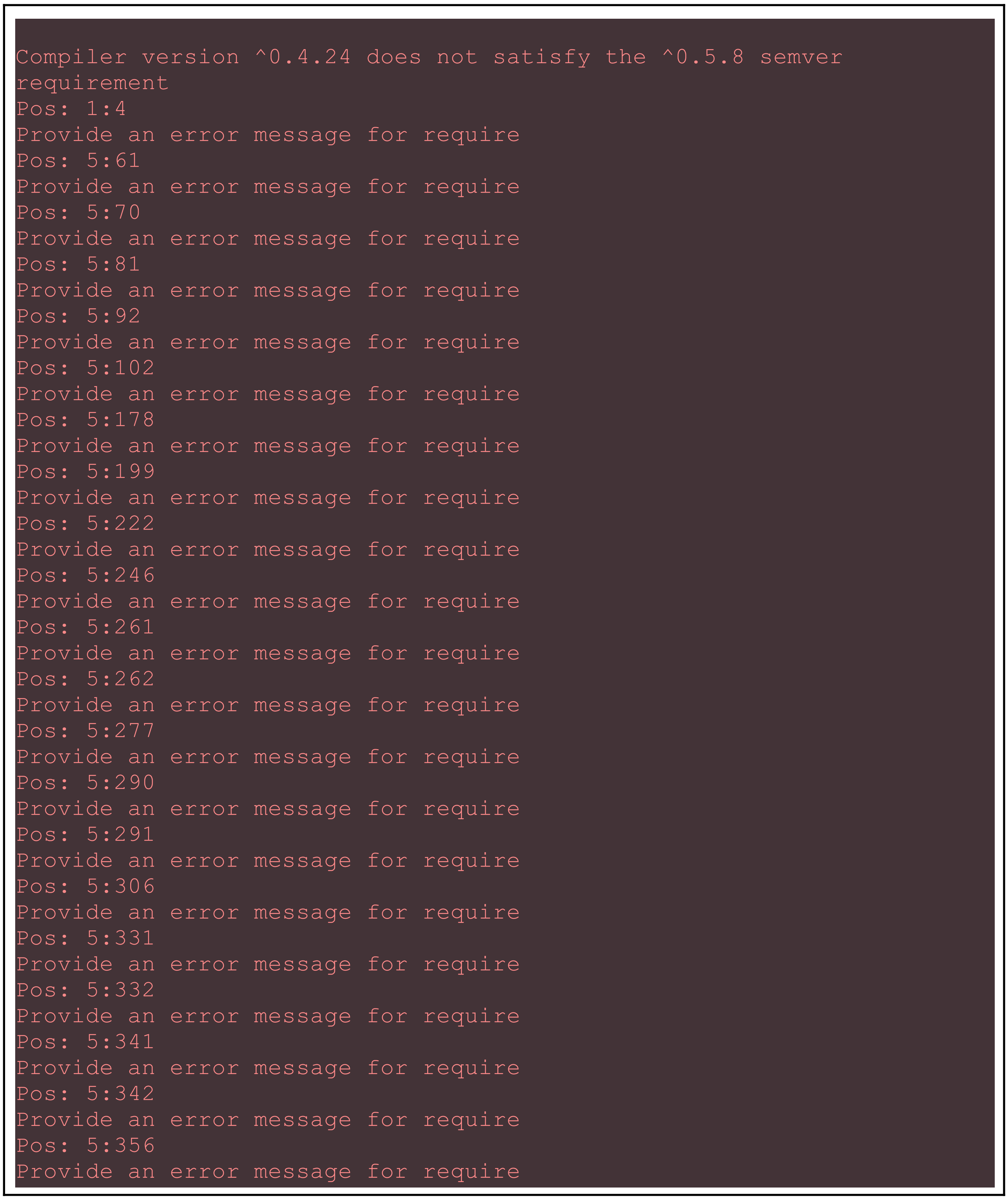
Data truncated:

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: 72:16:

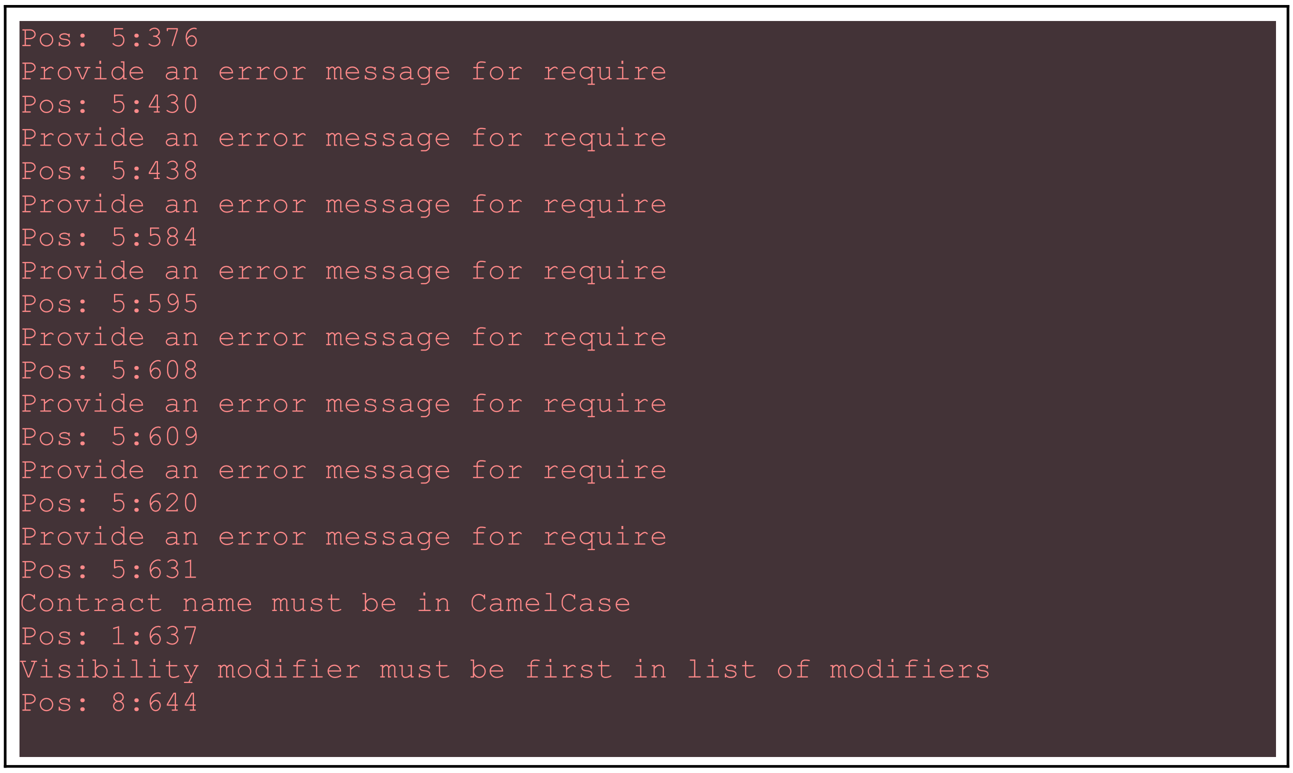
Solhint Linter

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.

chiliZ.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: 5:61
Provide an error message for require
Pos: 5:70
Provide an error message for require
Pos: 5:81
Provide an error message for require
Pos: 5:92
Provide an error message for require
Pos: 5:102
Provide an error message for require
Pos: 5:178
Provide an error message for require
Pos: 5:199
Provide an error message for require
Pos: 5:222
Provide an error message for require
Pos: 5:246
Provide an error message for require
Pos: 5:261
Provide an error message for require
Pos: 5:262
Provide an error message for require
Pos: 5:277
Provide an error message for require
Pos: 5:290
Provide an error message for require
Pos: 5:291
Provide an error message for require
Pos: 5:306
Provide an error message for require
Pos: 5:331
Provide an error message for require
Pos: 5:332
Provide an error message for require
Pos: 5:341
Provide an error message for require
Pos: 5:342
Provide an error message for require
Pos: 5:356
Provide an error message for require
```



```
Pos: 5:376
Provide an error message for require
Pos: 5:430
Provide an error message for require
Pos: 5:438
Provide an error message for require
Pos: 5:584
Provide an error message for require
Pos: 5:595
Provide an error message for require
Pos: 5:608
Provide an error message for require
Pos: 5:609
Provide an error message for require
Pos: 5:620
Provide an error message for require
Pos: 5:631
Contract name must be in CamelCase
Pos: 1:637
Visibility modifier must be first in list of modifiers
Pos: 8:644
```

Software analysis result:

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