SMART CONTRACT AUDIT REPORT

For

BITChro Token (BTCH)

http://bitchro.com/

Prepared By: SFI Team

Prepared for: BITChroTeam

Prepared on: 8/1/2022

Contract address:
0x881c3F97d994F7ba6B8a2d5488f32Ed9aB15e502

Table of Content

- Disclaimer
- Scope of the audit
- Check Vulnerabilities
- Issue Categories
- Issues Found – Code Review
- Source Lines
- Risk Level
- Capabilities
- Testing proves
- Inheritance graph
- Call Graph
- Source Units In Scope
- Unified Modeling Language (UML)
- Functions signature
- Automatic general report
- Summary of the audit

- ## Disclaimer

  This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.
  By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you.
  This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SaferICO) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

• Scope of the audit

The scope of this audit was to analyze and document the BITChro Token smart contract codebase for quality, security, and correctness.

• Introduction

During the period of **January 7, 2022, to January 8, 2022** - SaferICO

Team performed a security audit for **BITChro** smart contracts.

The project has 1 file. It contains approx 390 lines of Solidity code. Most of the functions and state variables are well commented on using the Nat spec documentation, but that does not create any vulnerability.

Source Code: https://bscscan.com/address/0x881c3f97d994f7ba6b8a2d5488f32ed9ab15e502#code

Check Vulnerabilities

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices automatically.

1. Unit tests passing.

2. Compilator warnings;

3. Race Conditions. Reentrancy. Cross-function Race Conditions. Pitfalls in Race Condition solutions;

4. Possible delays in data delivery;

5. Transaction-Ordering Dependence (front running);

6. Timestamp Dependence;

7. Integer Overflow and Underflow;

8. DoS with (unexpected) Revert;

9. DoS with Block Gas Limit

10. Call Depth Attack. Not relevant in modern ethereum network

11. Methods execution permissions;

12. Oracles calls;

13. Economy model. It’s important to forecast scenarios when a user is provided with additional economic motivation or faced with limitations. If application logic is based on incorrect economy model, the application will not function correctly and participants will incur financial losses. This type of issue is most often found in bonus rewards systems.

14. The impact of the exchange rate on the logic;

15. Private user data leaks.

• Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

- # Issues Found – Code Review

High severity issues

There are no High severity vulnerabilities found.

Medium severity issues

 # The owner can mint a new token anytime
Description
The owner has the ability to mint more token which can effect on the price of the token; this represents a risk for the users because in that case their funds will be more less in price.

Remediation
Make mint() function internal so no one can mint more tokens or they have to add max supply to mint function or use renounceOwnership function.

Status: Open.

Low severity issues

There are no Low severity vulnerabilities found .

Informational issues

#Missing SPDX-License-Identifier:

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information .

Remediation

Add License Identifier

```
// SPDX-License-Identifier: UNLICENSE
```

Status: Acknowledged

Constant calculations in the contract
Description
recalculated initialization will save 2847 units of gas in deployment

```
totalTokens = 1000000000  * 10 ** uint256(decimals());
```

Recommendation

Replace the initialization as

```
totalTokens = 1000000000000000000000000000;
```

Status: Acknowledged

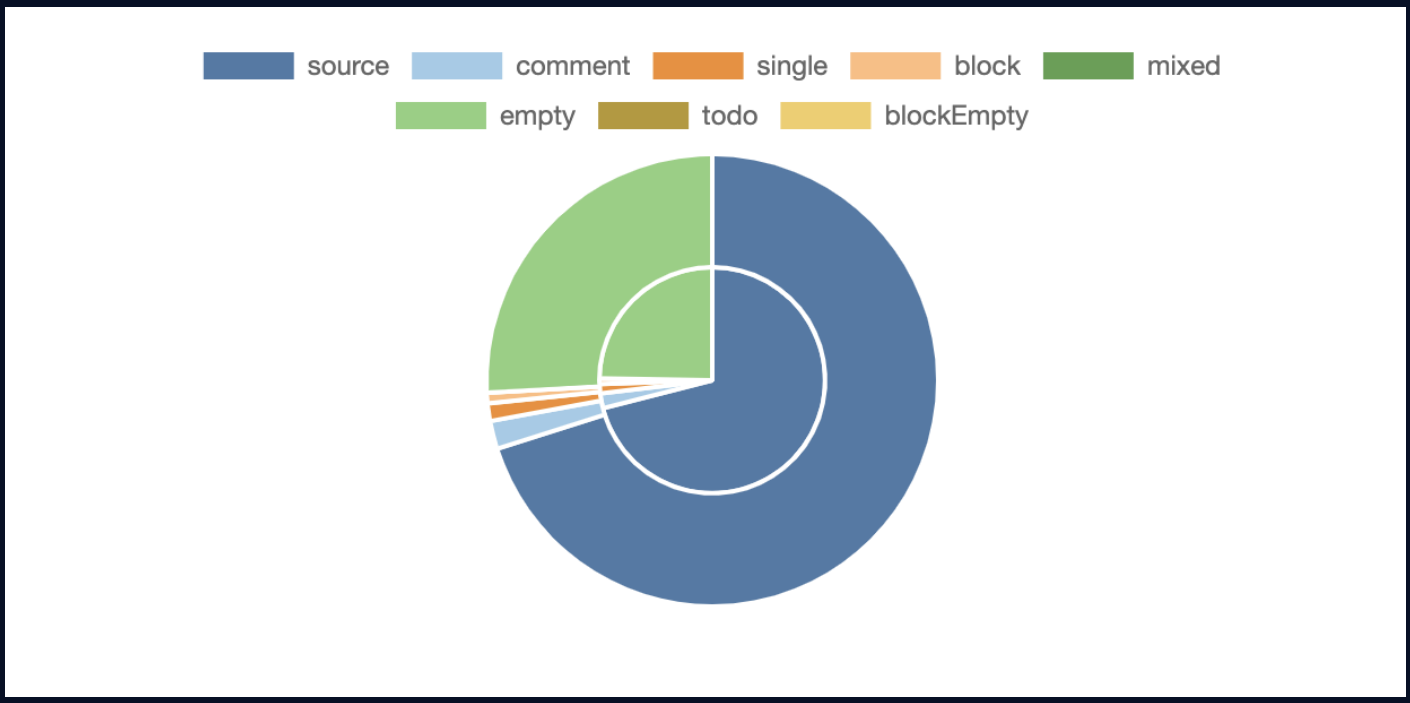
#Compiler version is old

Description

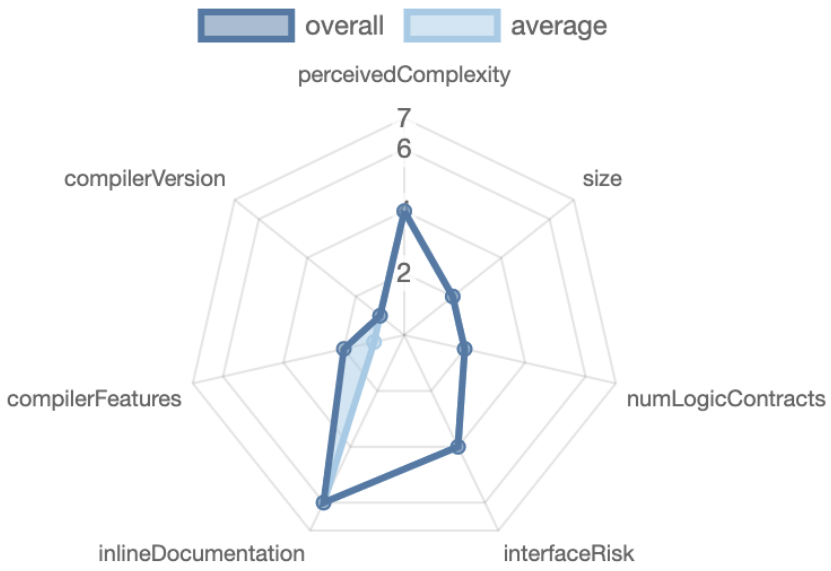
The compiler being used was released a 3 - 4 years ago. It's recommended to use more recent compiler version, there can be benefits like reduction in bytecode size etc.

Status: **Acknowledged**

- **Source Lines**

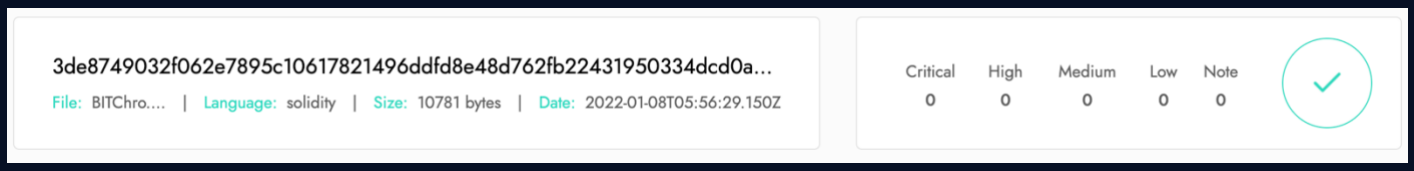


- **Risk Level**

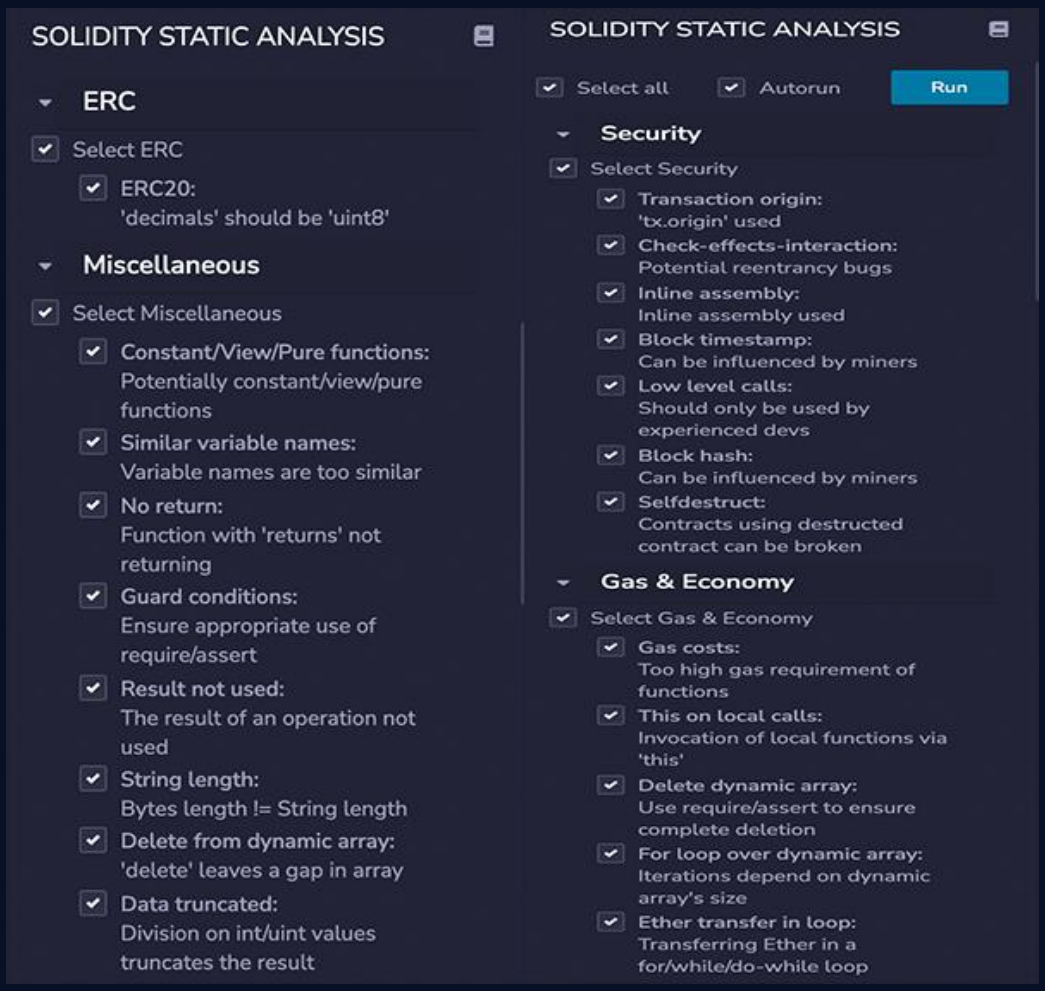


- **Testing proves**

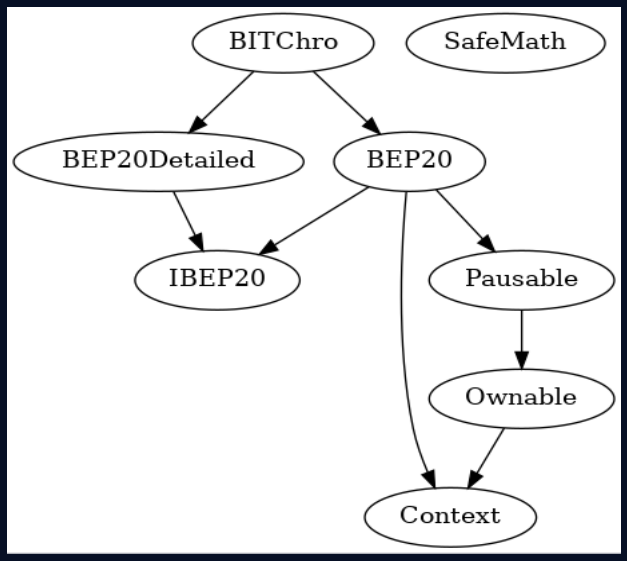
- **Check for security**



- **Solidity Static Analysis**

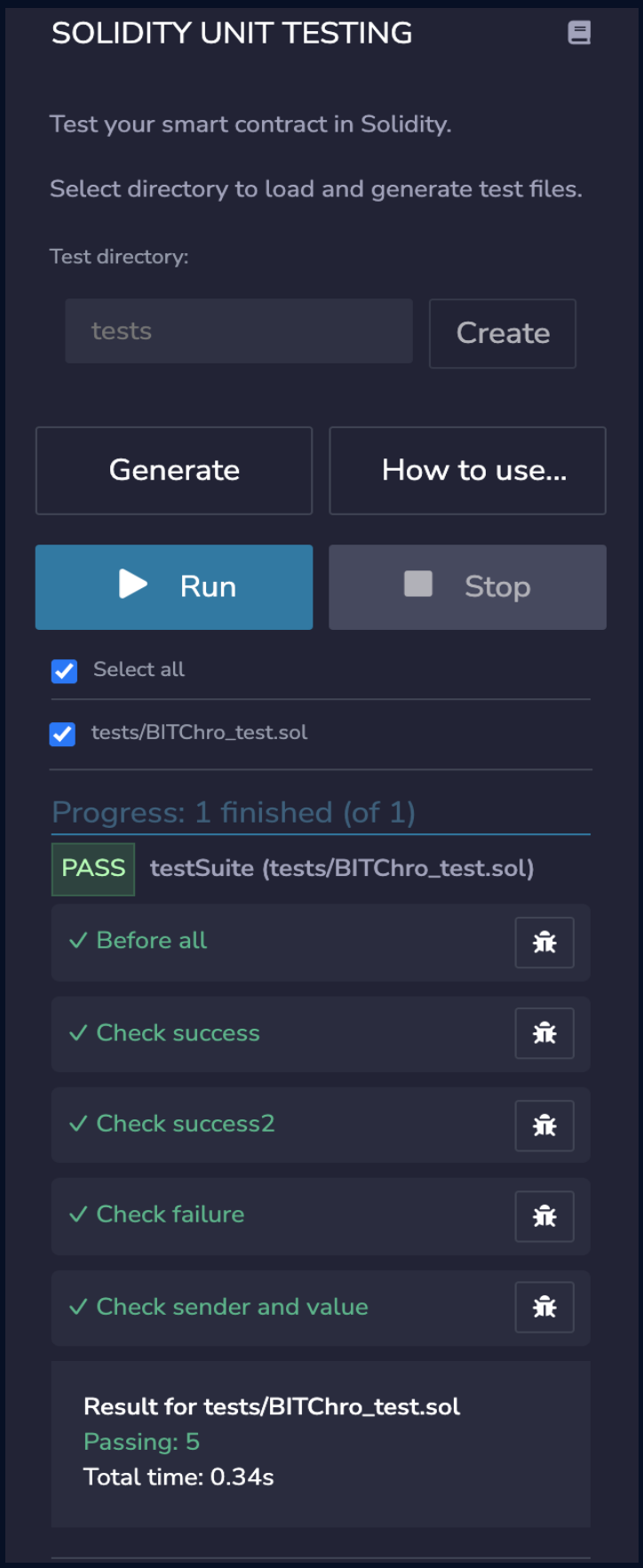


- ## Inheritance graph



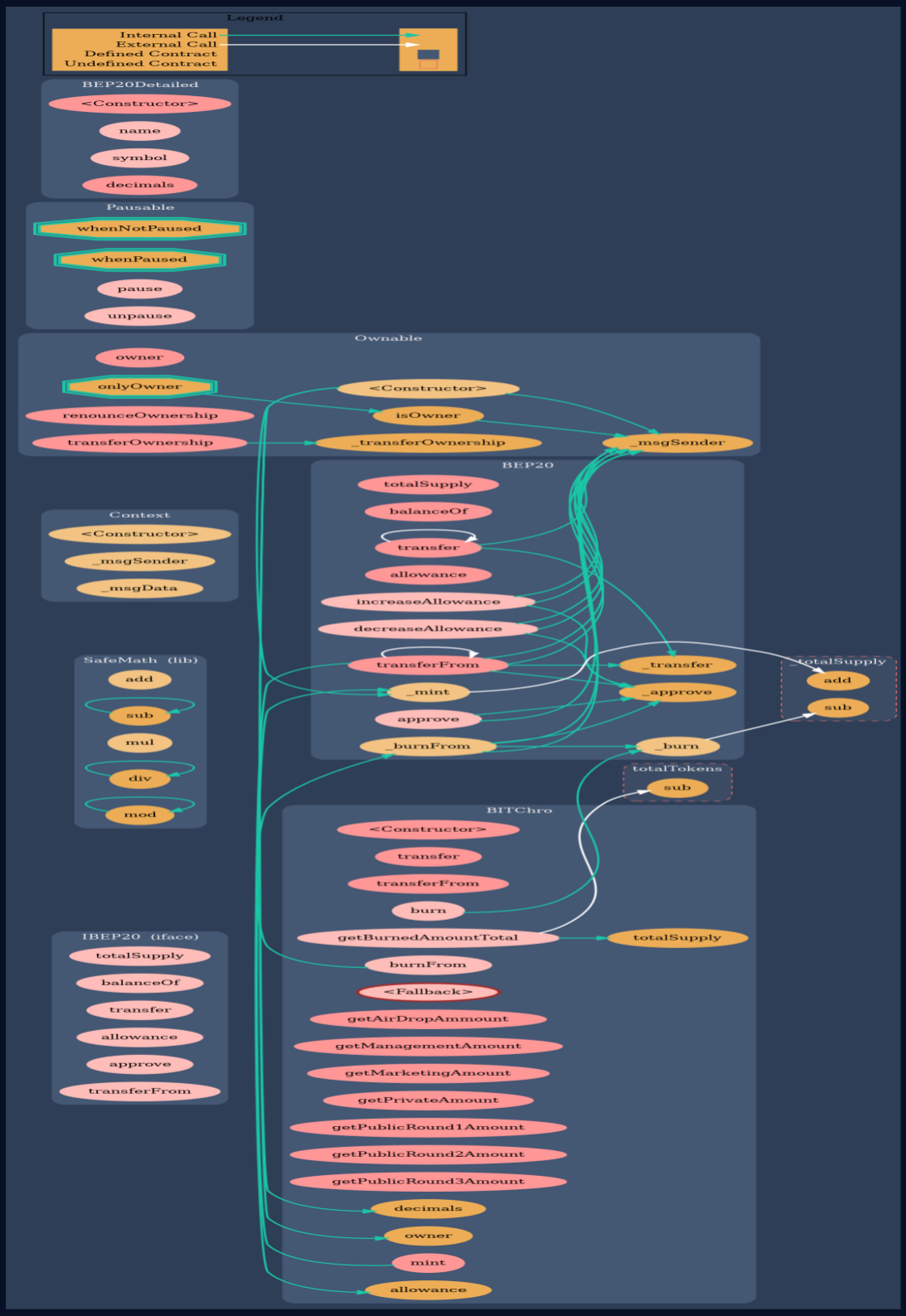
- ## Solidity Unit Testing Code & Results

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.22 <0.9.0;

// This import is automatically injected by Remix
import "remix_tests.sol";

// This import is required to use custom transaction context
// Although it may fail compilation in 'Solidity Compiler' plugin
// But it will work fine in 'Solidity Unit Testing' plugin
import "remix_accounts.sol";
import "../BITChro.sol";

// File name has to end with '_test.sol', this file can contain more than one testSuite contracts
contract testSuite {

    /// 'beforeAll' runs before all other tests
    /// More special functions are: 'beforeEach', 'beforeAll', 'afterEach' & 'afterAll'
```

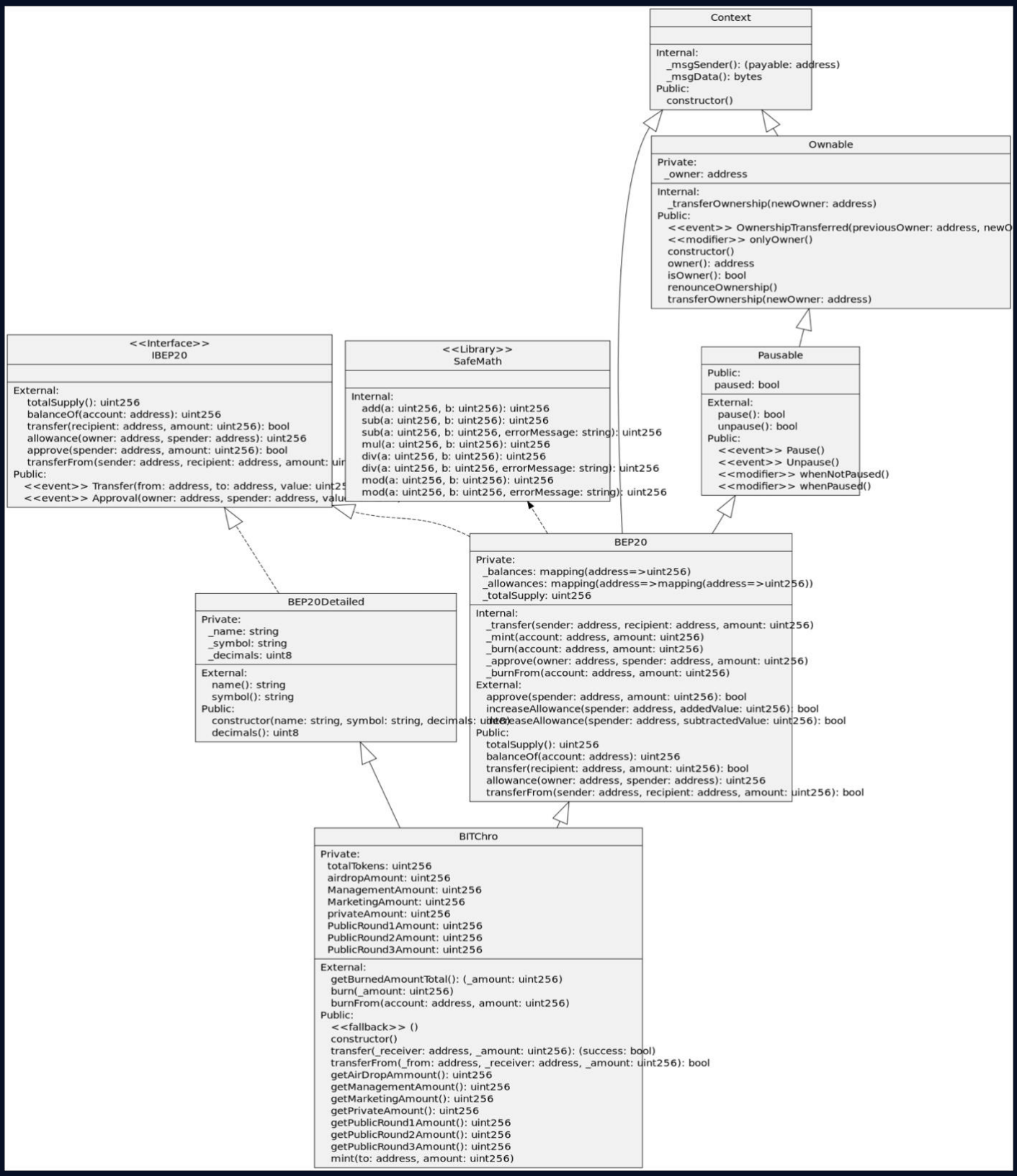



```
function beforeAll() public {
    // <instantiate contract>
    Assert.equal(uint(1), uint(1), "1 should be equal to 1");
}

function checkSuccess() public {
    // Use 'Assert' methods: https://remix-ide.readthedocs.io/en/latest/assert_library.html
    Assert.ok(2 == 2, 'should be true');
    Assert.greaterThan(uint(2), uint(1), "2 should be greater than to 1");
    Assert.lesserThan(uint(2), uint(3), "2 should be lesser than to 3");
}

function checkSuccess2() public pure returns (bool) {
    // Use the return value (true or false) to test the contract
    return true;
}

function checkFailure() public {
    Assert.notEqual(uint(1), uint(2), "1 should not be equal to 1");
}

/// Custom Transaction Context: https://remix-ide.readthedocs.io/en/latest/unittesting.html#customization
/// #sender: account-1
/// #value: 100
function checkSenderAndValue() public payable {
    // account index varies 0-9, value is in wei
    Assert.equal(msg.sender, TestsAccounts.getAccount(1), "Invalid sender");
    Assert.equal(msg.value, 100, "Invalid value");
}
}
```

- **Call Graph**


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BEP20Detailed
<Constructor>
name
symbol
decimals
Pausable
whenNotPaused
whenPaused
pause
unpause
Ownable
owner
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
isOwner
_transferOwnership
_msgSender
BEP20
totalSupply
balanceOf
transfer
allowance
increaseAllowance
decreaseAllowance
transferFrom
_mint
approve
_burnFrom
_transfer
_approve
_burn
_totalSupply
add
sub
totalTokens
sub
Context
<Constructor>
_msgSender
_msgData
SafeMath (lib)
add
sub
mul
div
mod
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
BITChro
<Constructor>
transfer
transferFrom
burn
getBurnedAmountTotal
totalSupply
burnFrom
<Fallback>
getAirDropAmmount
getManagementAmount
getMarketingAmount
getPrivateAmount
getPublicRound1Amount
getPublicRound2Amount
getPublicRound3Amount
decimals
owner
mint
allowance

- **Unified Modeling Language (UML)**



- ## Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
6	1	1	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
39	1

External	Internal	Private	Pure	View
17	45	0	8	21

StateVariables

Total	Public
16	1

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`0.5.17`		`yes`		

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`					

TryCatch	Σ Unchecked

- ## Source Units In Scope

Source Units in Scope

Source Units Analyzed: `1`
Source Units in Scope: `1` **(100%)**

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	BITChro.sol	7	1	390	368	267	8	226	
	Totals	**7**	**1**	**390**	**368**	**267**	**8**	**226**	

Legend: [−]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

- **Function Signature**

```
39509351 => increaseAllowance(address,uint256)
18160ddd => totalSupply()
70a08231 => balanceOf(address)
a9059cbb => transfer(address,uint256)
dd62ed3e => allowance(address,address)
095ea7b3 => approve(address,uint256)
23b872dd => transferFrom(address,address,uint256)
771602f7 => add(uint256,uint256)
b67d77c5 => sub(uint256,uint256)
e31bdc0a => sub(uint256,uint256,string)
c8a4ac9c => mul(uint256,uint256)
a391c15b => div(uint256,uint256)
b745d336 => div(uint256,uint256,string)
f43f523a => mod(uint256,uint256)
71af23e8 => mod(uint256,uint256,string)
119df25f => _msgSender()
8b49d47e => _msgData()
8da5cb5b => owner()
8f32d59b => isOwner()
715018a6 => renounceOwnership()
f2fde38b => transferOwnership(address)
d29d44ee => _transferOwnership(address)
8456cb59 => pause()
3f4ba83a => unpause()
a457c2d7 => decreaseAllowance(address,uint256)
30e0789e => _transfer(address,address,uint256)
4e6ec247 => _mint(address,uint256)
6161eb18 => _burn(address,uint256)
104e81ff => _approve(address,address,uint256)
a22b35ce => _burnFrom(address,uint256)
06fdde03 => name()
95d89b41 => symbol()
313ce567 => decimals()
1ec4280a => getBurnedAmountTotal()
42966c68 => burn(uint256)
```

```
79cc6790 => burnFrom(address,uint256)
9a575c37 => getAirDropAmmount()
6cced7b5 => getManagementAmount()
4d836455 => getMarketingAmount()
1c08d7da => getPrivateAmount()
8124ca2a => getPublicRound1Amount()
933ec353 => getPublicRound2Amount()
c7b887f0 => getPublicRound3Amount()
40c10f19 => mint(address,uint256)
```

- # Automatic General Report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/BITChro.sol	46c6c8509a31e2f8e1466ca5f40dfba08a308a47

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface				
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
SafeMath	Library				
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
Context	Implementation				
└	<Constructor>	Internal 🔒	🛑	
└	_msgSender	Internal 🔒		

| └ | _msgData | Internal 🔒 | | |

||||||

| **Ownable** | Implementation | Context |||

| └ | <Constructor> | Internal 🔒 | 🛑 | |

| └ | owner | Public ❗ | | NO❗ |

| └ | isOwner | Public ❗ | | NO❗ |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _transferOwnership | Internal 🔒 | 🛑 | |

||||||

| **Pausable** | Implementation | Ownable |||

| └ | pause | External ❗ | 🛑 | onlyOwner whenNotPaused |

| └ | unpause | External ❗ | 🛑 | onlyOwner whenPaused |

||||||

| **BEP20** | Implementation | Context, IBEP20, Pausable |||

| └ | totalSupply | Public ❗ | | NO❗ |

| └ | balanceOf | Public ❗ | | NO❗ |

| └ | transfer | Public ❗ | 🛑 | whenNotPaused |

| └ | allowance | Public ❗ | | NO❗ |

| └ | approve | External ❗ | 🛑 | whenNotPaused |

| └ | transferFrom | Public ❗ | 🛑 | whenNotPaused |

| └ | increaseAllowance | External ❗ | 🛑 | whenNotPaused |

| └ | decreaseAllowance | External ❗ | 🛑 | whenNotPaused |

| └ | _transfer | Internal 🔒 | 🛑 | |

| └ | _mint | Internal 🔒 | 🛑 | |

| └ | _burn | Internal 🔒 | 🛑 | |

| └ | _approve | Internal 🔒 | 🛑 | |

| └ | _burnFrom | Internal 🔒 | 🛑 | |

||||||

| **BEP20Detailed** | Implementation | IBEP20 |||

| └ | <Constructor> | Public ❗ | 🛑 | NO❗ |

| └ | name | External ❗ | | NO❗ |

| └ | symbol | External ❗ | | NO❗ |

| └ | decimals | Public ❗ | | NO❗ |

||||||

| **BITChro** | Implementation | BEP20Detailed, BEP20 |||

| └ | <Constructor> | Public ❗ | 🛑 | BEP20Detailed |

| └ | transfer | Public ❗ | 🛑 | NO❗ |

└	transferFrom	Public ❗	🛑	NO❗
└	getBurnedAmountTotal	External ❗		NO❗
└	burn	External ❗	🛑	NO❗
└	burnFrom	External ❗	🛑	NO❗
└	<Fallback>	External ❗	💷	NO❗
└	getAirDropAmmount	Public ❗		NO❗
└	getManagementAmount	Public ❗		NO❗
└	getMarketingAmount	Public ❗		NO❗
└	getPrivateAmount	Public ❗		NO❗
└	getPublicRound1Amount	Public ❗		NO❗
└	getPublicRound2Amount	Public ❗		NO❗
└	getPublicRound3Amount	Public ❗		NO❗
└	mint	Public ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💷	Function is payable

Summary of the Audit

According to all test, the customer`s solidity smart contract is Poor-**Secure**.

The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

The test found 0 critical, 0 high, 1 medium, 0 low issues, and 3 notes.