WhiteH2Coin

Smart Contract Audit Report

November 21, 2022

Introduction

1. About WhiteH2Coin

WHC, an international association of clean energy tech supporters, aims to accelerate and facilitate the mainstream adoption of LTC technology, reducing the carbon footprint and waste accumulation by transforming waste into white hydrogen - the new symbol for clean energy.

Visit https://www.whiteh2coin.com/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The Nitro League team has provided the following doc for audit:

1. https://files.whiteh2coin.com/Light_Paper_EN.pdf
2. https://files.whiteh2coin.com/WhiteH2Coin_Whitepaper_EN.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors, including -

1. Structural analysis of the smart contract is checked and verified.
2. An extensive automated testing of all the contracts under scope is conducted.
3. Line-by-line Manual Code review is conducted to evaluate, analyze and identify the potential security risks in the contract.
4. Evaluation of the contract's intended behavior and the documentation shared is imperative to verify the contract behaves as expected.
5. For complex and heavy contracts, adequate integration testing is conducted to ensure that contracts interact acceptably.
6. Storage layout verifications in the upgradeable contract are a must.
7. An important step in the audit procedure is highlighting and recommending better gas optimization techniques in the contract.

Audit Details

- Project Name: WhiteH2Coin
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github link: NA
- Contract name: Wh2c, forwarder, balance_checkler.sol
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The audit's focus was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level Reference

Every issue in this report were assigned a severity level from the following:

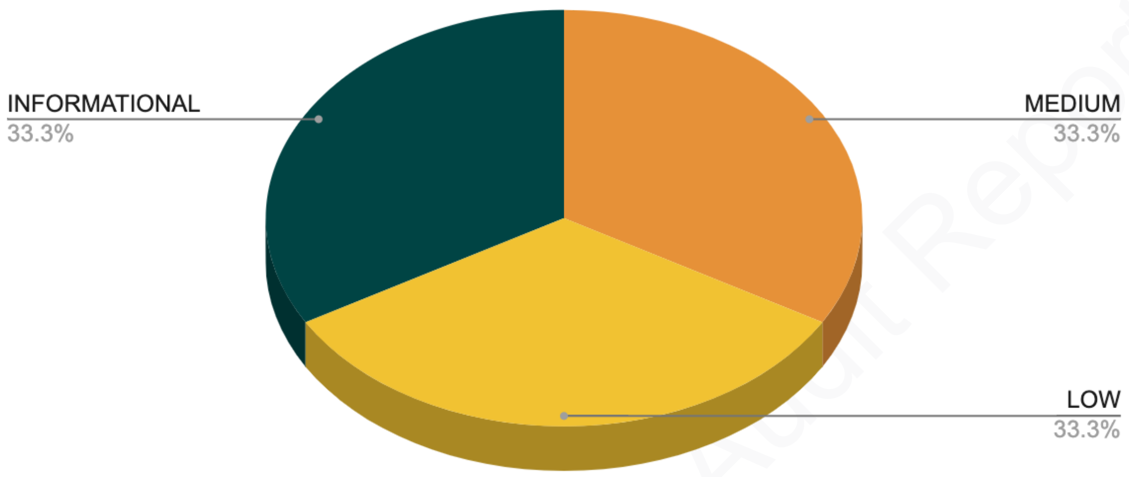
High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

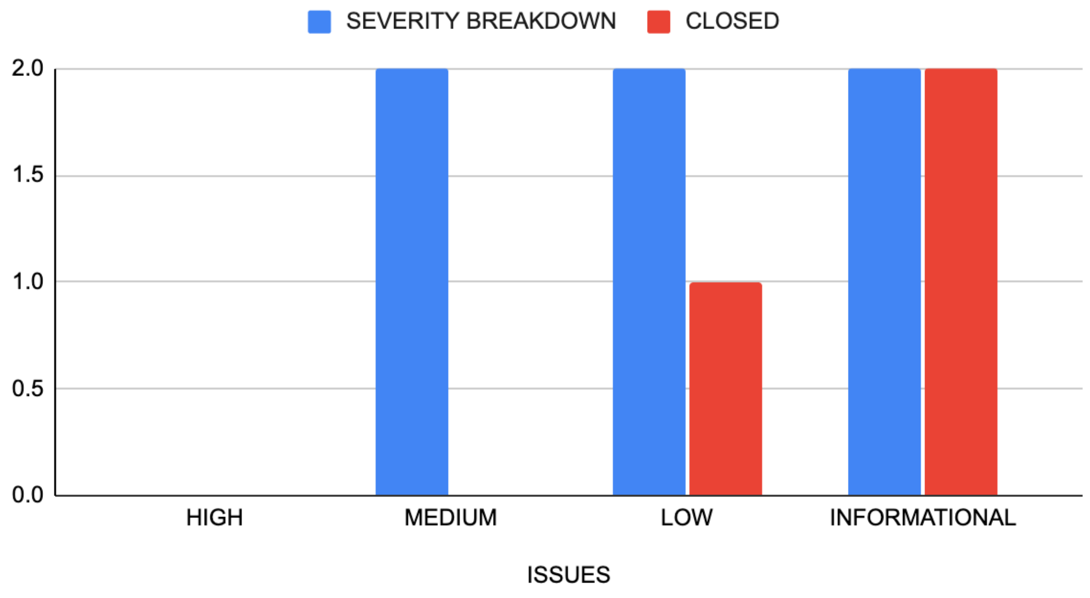
Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	-	2	1
Not Considered	-	-	1

SEVERITY



SEVERITY BREAKDOWN vs CLOSED



Finding Overview

#	Findings	Risk	Status
01	**balanceChecker.sol: No input validations to avoid array length mismatch**	Medium	Open
02	**balanceChecker.sol: checkBalances() function doesn't execute as intended**	Medium	Not Considered
03	**Forwarder.sol: No zero address validation**	Low	Closed
04	**WH2C.sol: Use of too many digits can be avoided**	Low	Not Considered
05	**Unlocked Pragma statements found in the contracts**	Informatory	Closed
06	**balanceChecker.sol: Coding Style Issues in the Contract**	Informatory	Closed

High Severity Issues

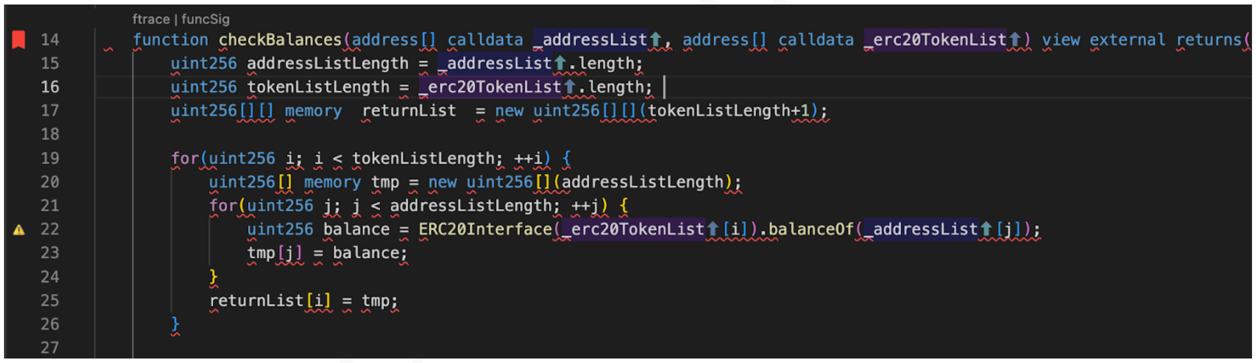
No issues were found.

Medium Severity Issues

1. **balanceChecker.sol: No input validations to avoid array length mismatch**
   Line no: 14-36

   Description:
   The checkBalances() function allows 2 different array arguments for the function where each of them goes through a for loop iteration.

   However, the function doesn't include any checkpoints to ensure that there is no array length mismatch while calling this function.



```
ftrace | funcSig
14  function checkBalances(address[] calldata _addressList, address[] calldata _erc20TokenList) view external returns(
15      uint256 addressListLength = _addressList.length;
16      uint256 tokenListLength = _erc20TokenList.length;
17      uint256[][] memory  returnList  = new uint256[][](tokenListLength+1);
18
19      for(uint256 i; i < tokenListLength; ++i) {
20          uint256[] memory tmp = new uint256[](addressListLength);
21          for(uint256 j; j < addressListLength; ++j) {
22              uint256 balance = ERC20Interface(_erc20TokenList[i]).balanceOf(_addressList[j]);
23              tmp[j] = balance;
24          }
25          returnList[i] = tmp;
26      }
27
```

   The absence of such validations might lead to unwanted function execution and is therefore not an adequate function design.

   Recommendation:
   Functions should include adequate require statements to validate the arguments being passed.

   Amended (November 21st, 2022): The team has fixed the issue, and it is no longer present in the code.

2. **balanceChecker.sol: checkBalances() function doesn't execute as intended**
   Line no: 14-36

   Description:
   During the review it was found that the checkBalances() function isn't designed in an adequate manner since the transaction reverts while checking the balance of addresses.
   The use of a 2-D array had also been included in an inappropriate meaning which results in a wrong execution of the function.

   Recommendation:
   It is recommended to redesign the function and includes specific test cases to verify the behavior. Additionally, the balance check of tokens and ETH value can either be included in separate functions or the use of arrays should be rechecked in the current function and required modifications should be made.

   Amended (November 21st, 2022): The team has fixed the issue, and it is no longer present in the code.

Low Severity Issues

1. **Forwarder.sol: No zero address validation**
   Line no- 36-38

   Description:
   The Forwarder contract includes the changeVault() function that updates an imperative address in the contract.

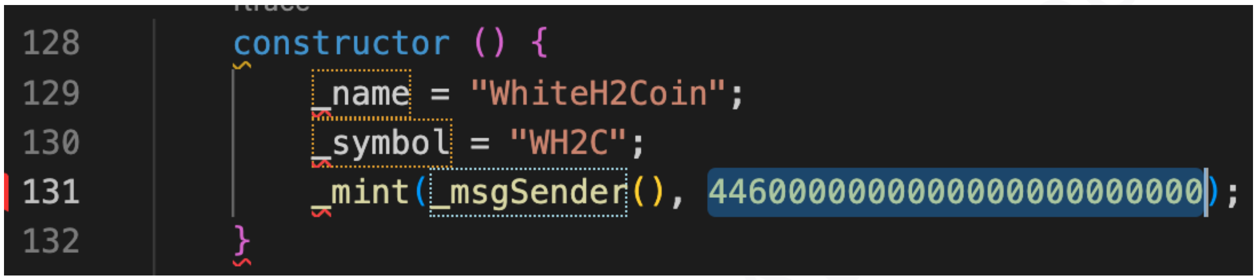
   However, during the automated testing of the contract it was found that no Zero Address Validation is implemented on the following functions while updating the address state variables of the contract:

   Recommendation:
   A require statement should be included in such functions to ensure no zero address is passed in the arguments.

   Amended (November 21st, 2022): The team has fixed the issue, and it is no longer present in the code.

2. **WH2C.sol: Use of too many digits can be avoided**
   Line no - 128-132

   Description:
   The above-mentioned lines have a large number of digits that makes it difficult to review and reduces the readability of the code.



```
128     constructor () {
129         _name = "WhiteH2Coin";
130         _symbol = "WH2C";
131         _mint(_msgSender(), 4460000000000000000000000);
132     }
```

   Recommendation:
   Ether Suffix could be used to symbolize the 10^18 zeros.

   Final Audit Comment: Not Considered

Recommendation / Informational

1. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

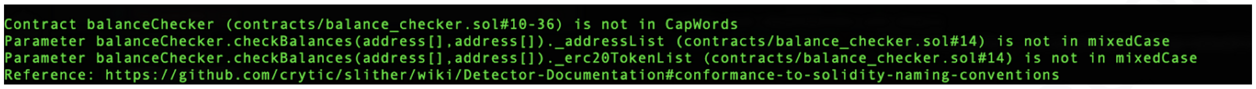
   It's not considered a better practice in Smart contract development to do so as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

   Amended (November 21st, 2022): The team has fixed the issue, and it is no longer present in the code.

2. balanceChecker.sol: Coding Style Issues in the Contract

Description
Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.



```
Contract balanceChecker (contracts/balance_checker.sol#10-36) is not in CapWords
Parameter balanceChecker.checkBalances(address[],address[])._addressList (contracts/balance_checker.sol#14) is not in mixedCase
Parameter balanceChecker.checkBalances(address[],address[])._erc20TokenList (contracts/balance_checker.sol#14) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

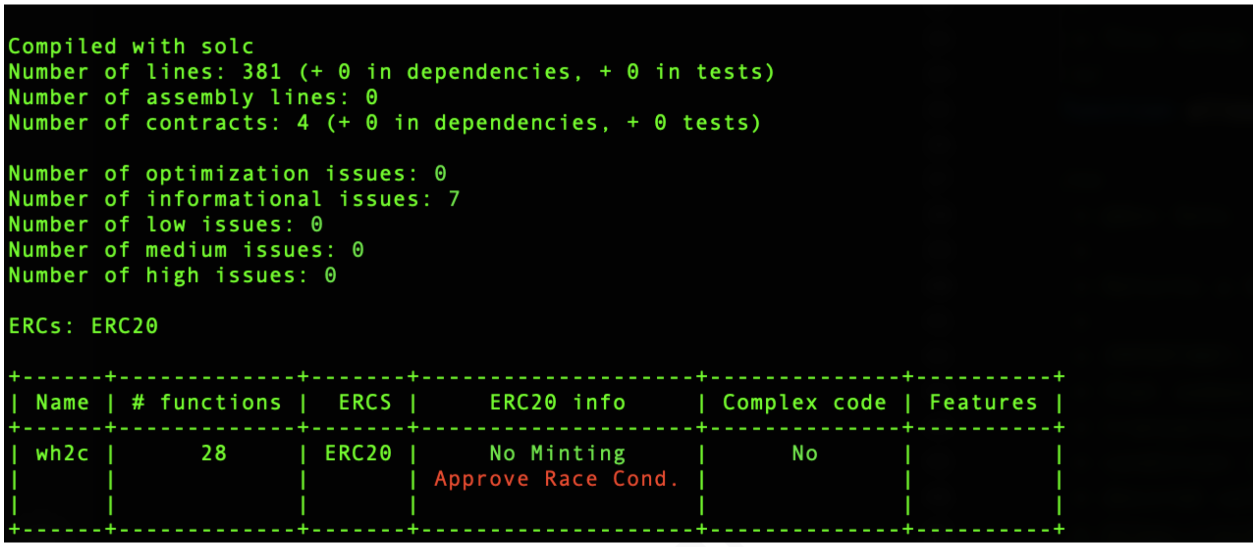
During the automated testing, it was found that the NAME contract had quite a few code-style issues. Please follow this link to find details on naming conventions in solidity code.

Recommendation:
Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Amended (November 21st, 2022): The team has fixed the issue, and it is no longer present in the code.

Automated Audit Result

1. WH2C.sol



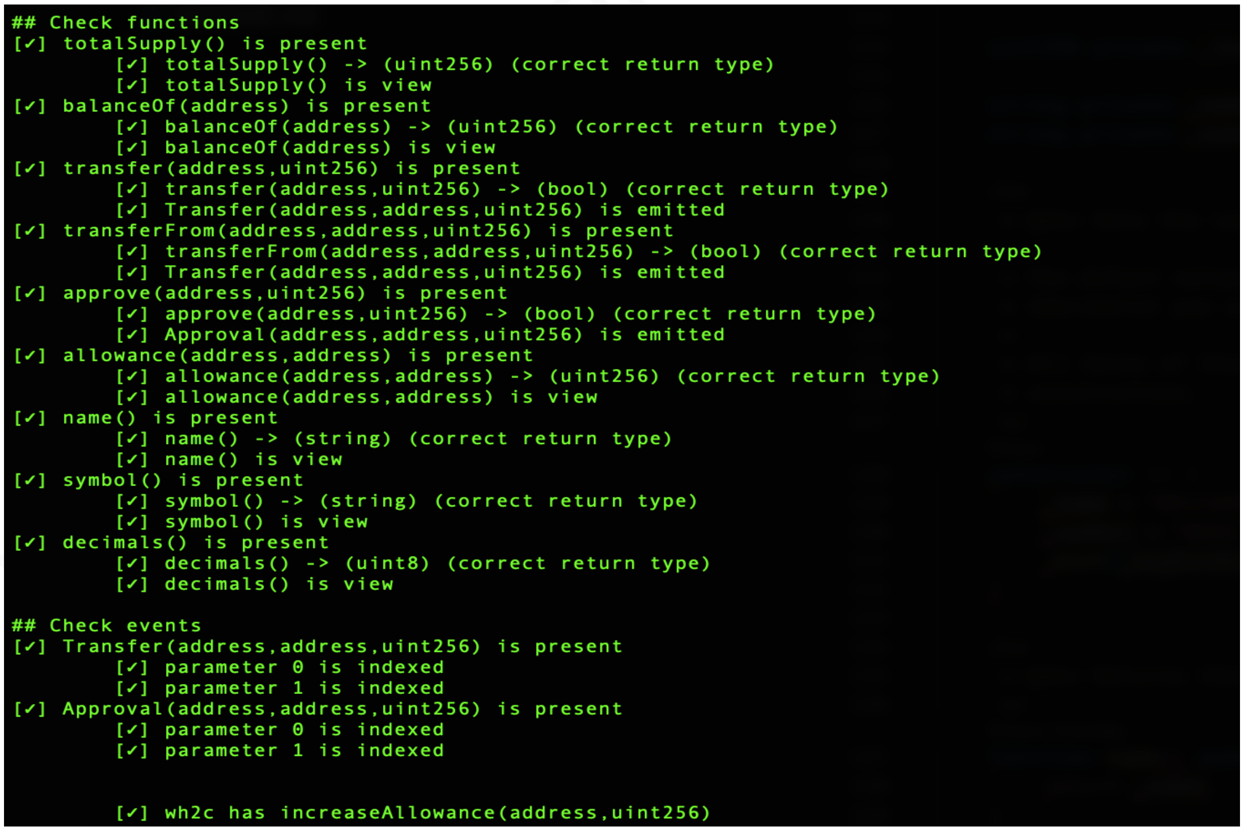
```
Compiled with solc
Number of lines: 381 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 4 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 7
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+------+-------------+-------+--------------------+--------------+----------+
| Name | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+------+-------------+-------+--------------------+--------------+----------+
| wh2c |      28     | ERC20 |     No Minting     |      No      |          |
|      |             |       | Approve Race Cond. |              |          |
|      |             |       |                    |              |          |
+------+-------------+-------+--------------------+--------------+----------+
```

2. ERC20 Compliance Check

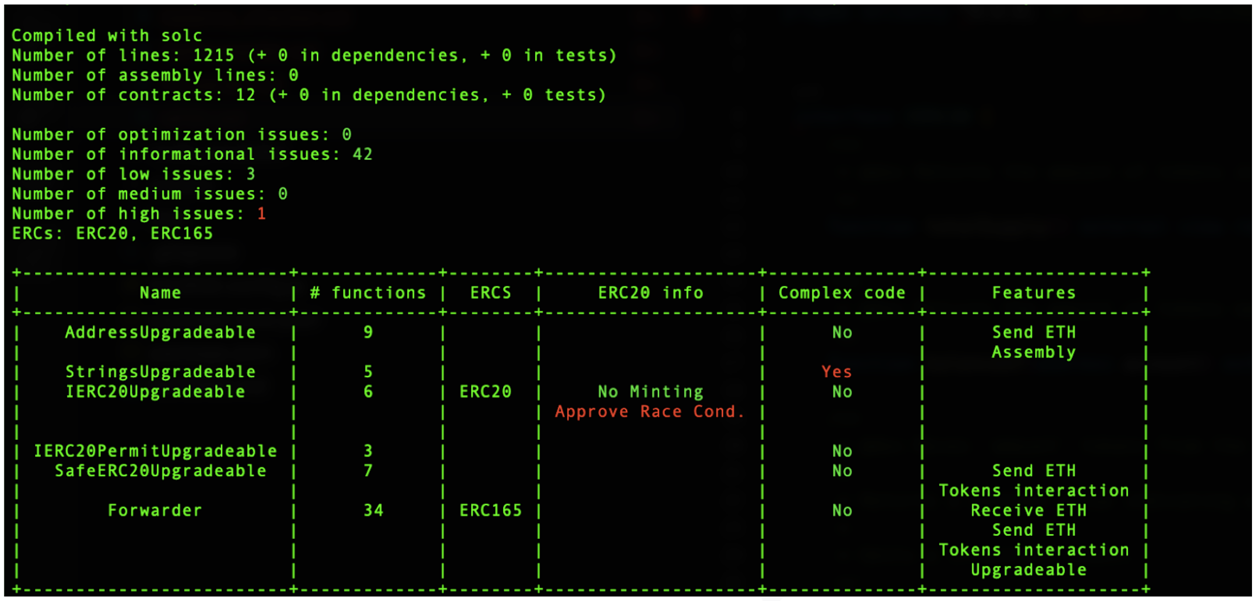


```
## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> (uint256) (correct return type)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> (uint256) (correct return type)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> (bool) (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> (bool) (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> (bool) (correct return type)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> (uint256) (correct return type)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> (string) (correct return type)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> (string) (correct return type)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> (uint8) (correct return type)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] wh2c has increaseAllowance(address,uint256)
```

3. Forwarder.sol

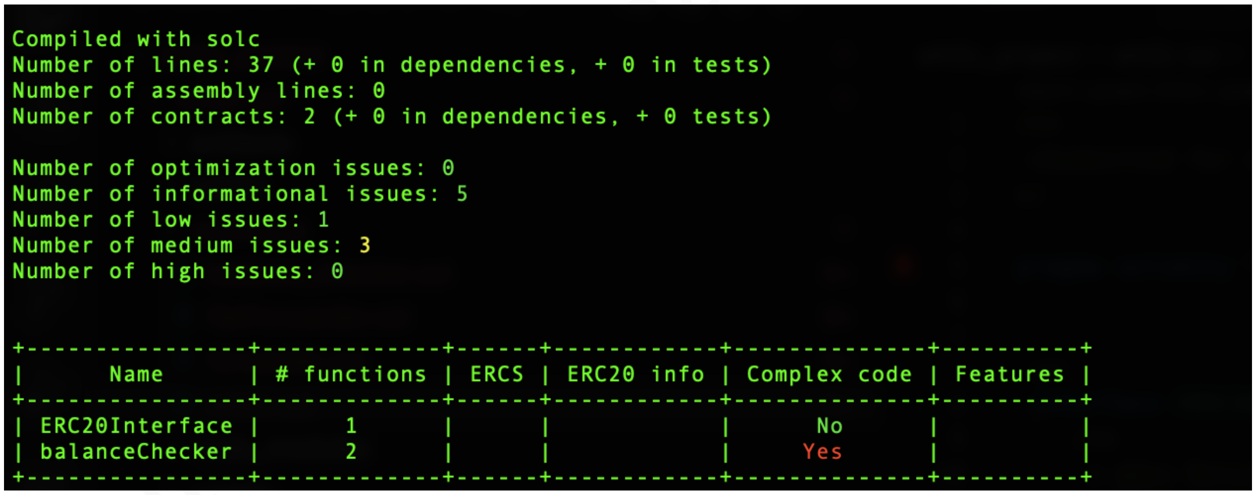


```
Compiled with solc
Number of lines: 1215 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 12 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 42
Number of low issues: 3
Number of medium issues: 0
Number of high issues: 1
ERCs: ERC20, ERC165

+-----------------------+------------+-------+------------------+-------------+-------------------+
|          Name         | # functions |  ERCS |    ERC20 info    | Complex code |      Features     |
+-----------------------+------------+-------+------------------+-------------+-------------------+
|    AddressUpgradeable  |      9     |       |                  |      No     |      Send ETH     |
|                       |            |       |                  |             |      Assembly     |
|    StringsUpgradeable  |      5     |       |                  |     Yes     |                   |
|    IERC20Upgradeable   |      6     | ERC20 |    No Minting    |      No     |                   |
|                       |            |       | Approve Race Cond. |           |                   |
|                       |            |       |                  |             |                   |
| IERC20PermitUpgradeable |    3     |       |                  |      No     |                   |
|   SafeERC20Upgradeable |      7     |       |                  |      No     |      Send ETH     |
|                       |            |       |                  |             | Tokens interaction |
|       Forwarder       |     34     | ERC165 |                 |      No     |    Receive ETH    |
|                       |            |       |                  |             |      Send ETH     |
|                       |            |       |                  |             | Tokens interaction |
|                       |            |       |                  |             |    Upgradeable    |
+-----------------------+------------+-------+------------------+-------------+-------------------+
```

4. Balance Checker



```
Compiled with solc
Number of lines: 37 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 5
Number of low issues: 1
Number of medium issues: 3
Number of high issues: 0


+----------------+-------------+------+------------+--------------+----------+
|      Name      | # functions | ERCS | ERC20 info | Complex code | Features |
+----------------+-------------+------+------------+--------------+----------+
| ERC20Interface |      1      |      |            |      No      |          |
| balanceChecker |      2      |      |            |     Yes      |          |
+----------------+-------------+------+------------+--------------+----------+
```

Concluding Remarks

While conducting the audits of the WhiteH2Coin smart contracts, it was observed that the contracts contain Medium and Low severity issues.

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the WhiteH2Coin platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

ImmuneBytes