SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

For

ARABELLA

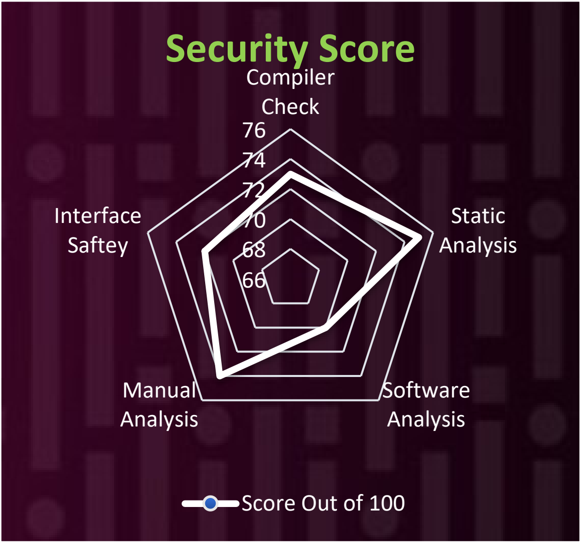
Table of Contents

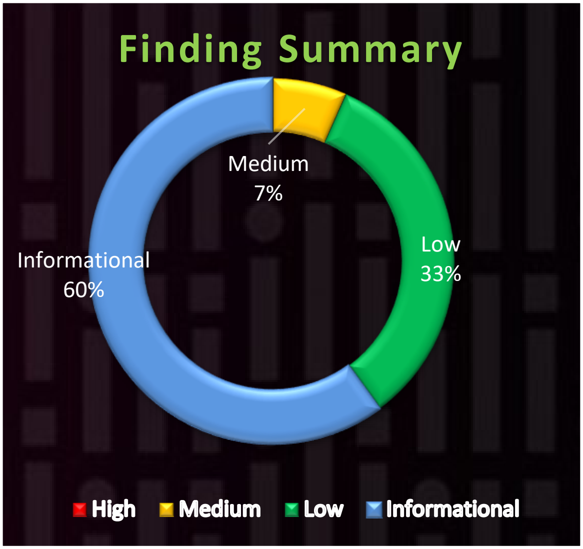
Disclaimer

The Secureverse team examined this smart contract in accordance with industry best practices. We made every effort to secure the code and provide this report. audits done by smart contract auditors and automated algorithms; however, it is crucial to remember that you should not rely entirely on this report. The smart contract may have flaws that allow for hacking. As a result, the audit cannot ensure the explicit security of the audited smart contracts. The Secureverse and its audit report do not encourage readers to consider them as providing any project-related financial or legal advice.

Executive Summary

Project Name	ABLC
Project Type	DeFi
Audit Scope	Check Security and code quality
Audit Method	Static and Manual
Audit Timeline	6th Oct 2022 to 7th Oct 2022
Source Code	ABLC.sol
Source code Hash	5816eea3c57bee49d8aeefab18dcc85dbd51f5c482f1b78e102f9e20175cc730





Issue Tracking Table

	High	Medium	Low	Informational
Open Issues	-	-	-	-
Acknowledged Issues	-	1	5	9
Resolved Issues	-	-	-	-

Types of Severities

- High: The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- Medium: The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
- Low: The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
- Informational: The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth.

Types of Issues

- Open: Security vulnerabilities identified that must be resolved and are currently unresolved.
- Acknowledged: Vulnerabilities which have been acknowledged but are yet to be resolved.
- Resolved: These are the issues identified in the initial audit and have been successfully fixed.

n

Checked Vulnerabilities

- Re-entrancy
- Access control
- Denial of service
- Timestamp Dependence
- Integer overflow/Underflow
- Transaction Order Dependency
- Requirement Violation
- Functions Visibility Check
- Mathematical calculations
- Dangerous strict equalities
- Unchecked Return values
- Hard coded information
- Safe Ether Transfer
- Gas Consumption
- Incorrect Inheritance Order
- Centralization
- Unsafe external calls
- Business logic and specification
- Input validation
- Incorrect Modifier
- Missing events
- Assembly usage
- ERC777 hooks
- Token handling

Methods

Audit at Secureverse is performed by the experts and they make sure that audited project must comply with the industry security standards.

Secureverse audit methodology includes following key:

- In depth review of the white paper
- In depth analysis of project and code documentation.
- Checking the industry standards used in Code/Project.
- Checking and understanding Core Functionality of the Code.
- Comparing the code with documentation.
- Static analysis of the code.
- Manual analysis of the code.
- Gas Optimization and Function Testing.
- Verification of the overall audit.
- Report writing.

The following techniques, methods and tools were used to review all the smart contracts.

- Static Analysis

Static analysis has been done by using the open source and state of the art automatic smart contract vulnerability scanning tools.

- Manual Analysis

Manual analysis is done by our smart contract auditors' team by performing in depth analysis of the smart contract and identify potential vulnerabilities. Auditor also review and verify all the static analysis results to prevent the false positives identified by automated tools.

- Gas Consumption and Function Testing

Function testing done by auditors by manually writing customized test cases for the smart contract to verify the intended behavior as per code and documentation. Gas Optimization done by reviews potential gas consumption by contract in production.

- Tools and Platforms used for Audit
    - Remix IDE
    - Hardhat
    - Mythril
    - Truffle Team
    - Solhint
    - Solidityscan.com
    - Slither
    - Consensys Surya
    - Open Zeppelin Code Analyzer
    - Manticore

Low Severity Issues:

Vulnerability:	**For Identifying Contract, code depends on Contract size**
Reference:	https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol#L290-L296
Description:	For checking an address is contract or EOA, Address library is checking the code size associated with that address, it assumes that if code size 0 then address is EOA, but it's possible that a contract could have zero code size.
Remediation:	One remediation come to mind is that check for tx.origin and msg.sender, if tx.origin == msg.sender then it's a normal EOA, it's a simple way to prevent bots.
Status:	Acknowledged

Vulnerability:	**Contract Used Openzeppelin Ownable contract, So change, Owner should be a 2-step process**
Reference:	https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol#L550-L555
Description:	This contract inherits from OpenZeppelin's library and the transferOwnership() function is the default one (a one-step process). It's possible that the onlyOwner role mistakenly transfers ownership to a wrong address, resulting in a loss of the onlyOwner role.
Remediation:	Consider overriding the default transferOwnership() function to first nominate an address as the pending owner and implementing an acceptOwnership() function which is called by the pending owner to confirm the transfer.
Status:	Acknowledged

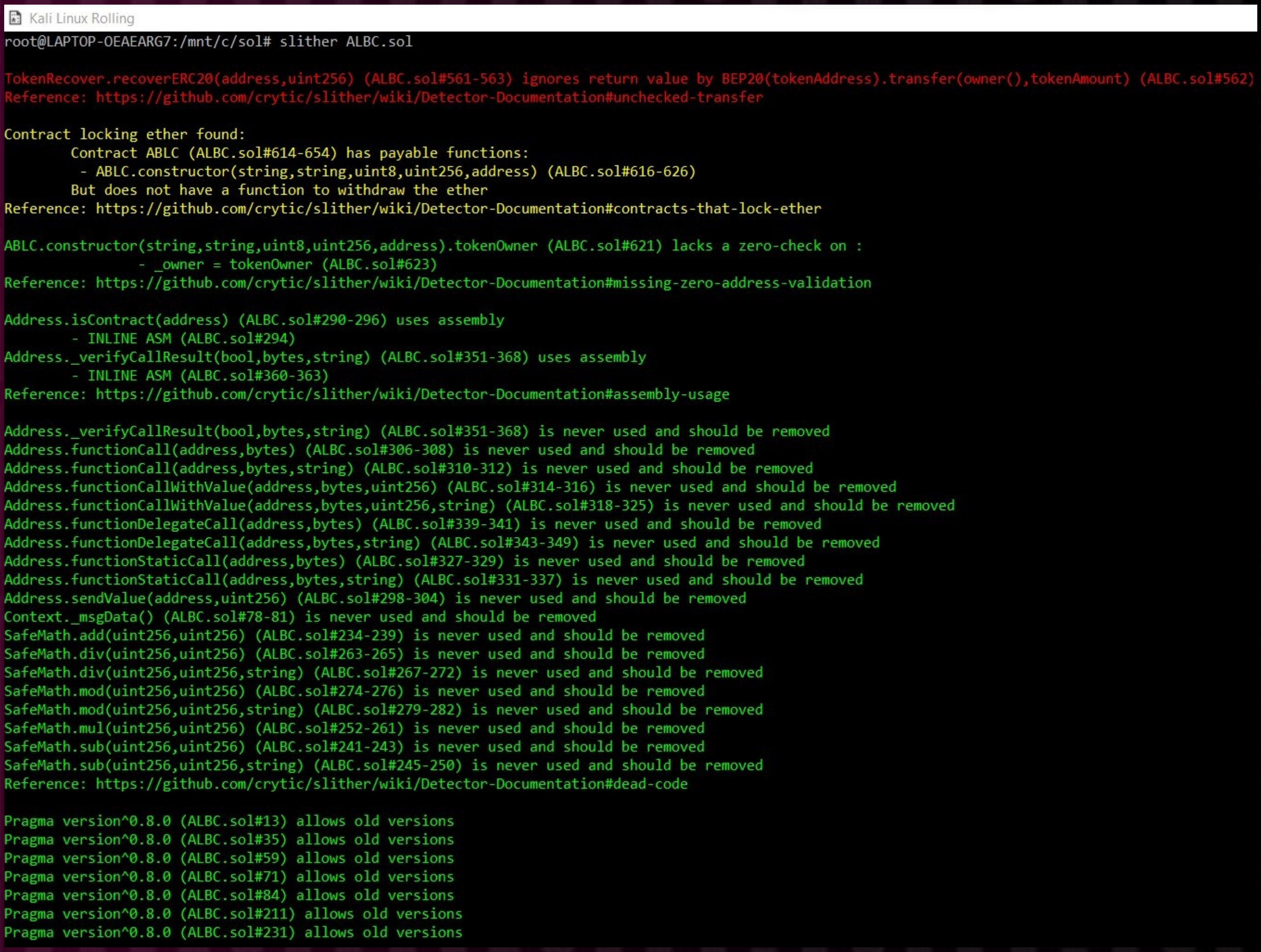
Vulnerability:	**Should be an extra layer security above renounceOwnership() from Ownable contract**
Reference:	https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol#L245-L248
Description:	This may possible that owner by mistake call renounceOwnership() function that lead to set owner to zero address, This will be dangerous situation like this type of contract, it recommended to make an additional layer of security above this function.
Remediation:	May implement a 2-step process on it. Like their will be a boon, in setRenounceOwner() it will set bool to true, and in executeRenounceOwner() it first check bool is true or not then set owner to Zero address. Obviously these 2 functions should be onlyOwner function.
Status:	Acknowledged

Vulnerability:	**A Blacklist user can't send Token, but it can receive Token**
Reference:	https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol#L122-L125 https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol#L136-L145
Description:	Due to absence of proper documentation, it is not sure it's a bug or feature. There is no check of blacklist for Recipient address in both transfer and transferFrom function.
Remediation:	If it's a bug then should be a check for recipient is blacklisted or not.
Status:	Acknowledged

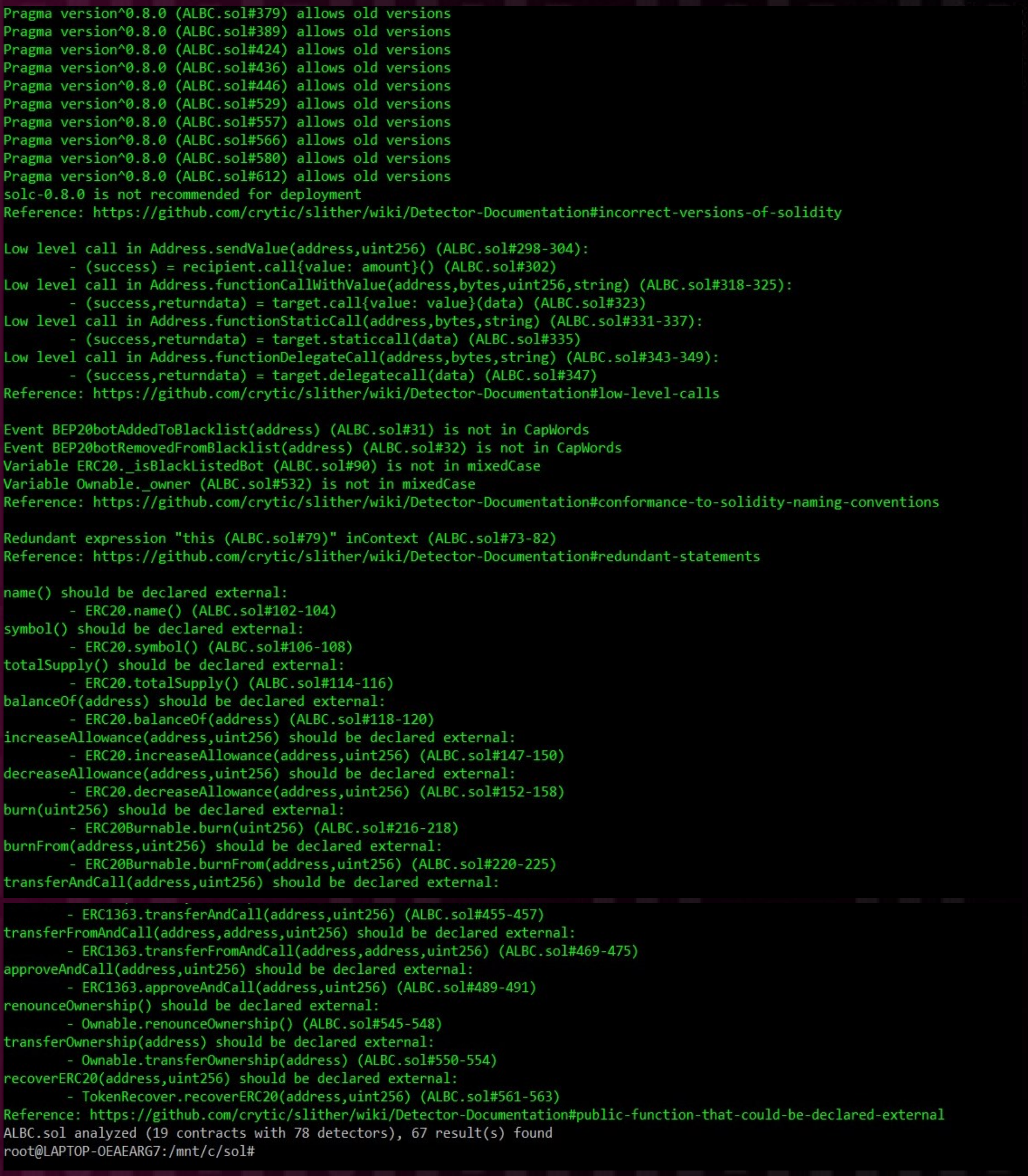
Vulnerability:	Unlocked Pragma Used
Reference:	https://github.com/LBM-Blockchain-Solution/ABLC/blob/main/ABLC.sol
Description:	Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.
Remediation:	Should lock pragmas to a specific compiler version.
Status:	Acknowledged

Automated Analysis

- Slither:
  - Issue Found: Total 67 issues were prompted in the contracts and most of them was found false positive.
  - The True positive issue is already covered in manual analysis.



```
root@LAPTOP-OEAEARG7:/mnt/c/sol# slither ALBC.sol

TokenRecover.recoverERC20(address,uint256) (ALBC.sol#561-563) ignores return value by BEP20(tokenAddress).transfer(owner(),tokenAmount) (ALBC.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Contract locking ether found:
        Contract ABLC (ALBC.sol#614-654) has payable functions:
         - ABLC.constructor(string,string,uint8,uint256,address) (ALBC.sol#616-626)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

ABLC.constructor(string,string,uint8,uint256,address).tokenOwner (ALBC.sol#621) lacks a zero-check on :
                - _owner = tokenOwner (ALBC.sol#623)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Address.isContract(address) (ALBC.sol#290-296) uses assembly
        - INLINE ASM (ALBC.sol#294)
Address._verifyCallResult(bool,bytes,string) (ALBC.sol#351-368) uses assembly
        - INLINE ASM (ALBC.sol#360-363)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._verifyCallResult(bool,bytes,string) (ALBC.sol#351-368) is never used and should be removed
Address.functionCall(address,bytes) (ALBC.sol#306-308) is never used and should be removed
Address.functionCall(address,bytes,string) (ALBC.sol#310-312) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (ALBC.sol#314-316) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (ALBC.sol#318-325) is never used and should be removed
Address.functionDelegateCall(address,bytes) (ALBC.sol#339-341) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (ALBC.sol#343-349) is never used and should be removed
Address.functionStaticCall(address,bytes) (ALBC.sol#327-329) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (ALBC.sol#331-337) is never used and should be removed
Address.sendValue(address,uint256) (ALBC.sol#298-304) is never used and should be removed
Context._msgData() (ALBC.sol#78-81) is never used and should be removed
SafeMath.add(uint256,uint256) (ALBC.sol#234-239) is never used and should be removed
SafeMath.div(uint256,uint256) (ALBC.sol#263-265) is never used and should be removed
SafeMath.div(uint256,uint256,string) (ALBC.sol#267-272) is never used and should be removed
SafeMath.mod(uint256,uint256) (ALBC.sol#274-276) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (ALBC.sol#279-282) is never used and should be removed
SafeMath.mul(uint256,uint256) (ALBC.sol#252-261) is never used and should be removed
SafeMath.sub(uint256,uint256) (ALBC.sol#241-243) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (ALBC.sol#245-250) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (ALBC.sol#13) allows old versions
Pragma version^0.8.0 (ALBC.sol#35) allows old versions
Pragma version^0.8.0 (ALBC.sol#59) allows old versions
Pragma version^0.8.0 (ALBC.sol#71) allows old versions
Pragma version^0.8.0 (ALBC.sol#84) allows old versions
Pragma version^0.8.0 (ALBC.sol#211) allows old versions
Pragma version^0.8.0 (ALBC.sol#231) allows old versions
```



```
Pragma version^0.8.0 (ALBC.sol#379) allows old versions
Pragma version^0.8.0 (ALBC.sol#389) allows old versions
Pragma version^0.8.0 (ALBC.sol#424) allows old versions
Pragma version^0.8.0 (ALBC.sol#436) allows old versions
Pragma version^0.8.0 (ALBC.sol#446) allows old versions
Pragma version^0.8.0 (ALBC.sol#529) allows old versions
Pragma version^0.8.0 (ALBC.sol#557) allows old versions
Pragma version^0.8.0 (ALBC.sol#566) allows old versions
Pragma version^0.8.0 (ALBC.sol#580) allows old versions
Pragma version^0.8.0 (ALBC.sol#612) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (ALBC.sol#298-304):
        - (success) = recipient.call{value: amount}() (ALBC.sol#302)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (ALBC.sol#318-325):
        - (success,returndata) = target.call{value: value}(data) (ALBC.sol#323)
Low level call in Address.functionStaticCall(address,bytes,string) (ALBC.sol#331-337):
        - (success,returndata) = target.staticcall(data) (ALBC.sol#335)
Low level call in Address.functionDelegateCall(address,bytes,string) (ALBC.sol#343-349):
        - (success,returndata) = target.delegatecall(data) (ALBC.sol#347)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Event BEP20botAddedToBlacklist(address) (ALBC.sol#31) is not in CapWords
Event BEP20botRemovedFromBlacklist(address) (ALBC.sol#32) is not in CapWords
Variable ERC20._isBlackListedBot (ALBC.sol#90) is not in mixedCase
Variable Ownable._owner (ALBC.sol#532) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (ALBC.sol#79)" inContext (ALBC.sol#73-82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

name() should be declared external:
        - ERC20.name() (ALBC.sol#102-104)
symbol() should be declared external:
        - ERC20.symbol() (ALBC.sol#106-108)
totalSupply() should be declared external:
        - ERC20.totalSupply() (ALBC.sol#114-116)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (ALBC.sol#118-120)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (ALBC.sol#147-150)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (ALBC.sol#152-158)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (ALBC.sol#216-218)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (ALBC.sol#220-225)
transferAndCall(address,uint256) should be declared external:
        - ERC1363.transferAndCall(address,uint256) (ALBC.sol#455-457)
transferFromAndCall(address,address,uint256) should be declared external:
        - ERC1363.transferFromAndCall(address,address,uint256) (ALBC.sol#469-475)
approveAndCall(address,uint256) should be declared external:
        - ERC1363.approveAndCall(address,uint256) (ALBC.sol#489-491)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (ALBC.sol#545-548)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (ALBC.sol#550-554)
recoverERC20(address,uint256) should be declared external:
        - TokenRecover.recoverERC20(address,uint256) (ALBC.sol#561-563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
ALBC.sol analyzed (19 contracts with 78 detectors), 67 result(s) found
root@LAPTOP-OEAEARG7:/mnt/c/sol#
```

About Secureverse

Secureverse is the Singapore and India based emerging Web3 Security solution provider. We at Secureverse provides the Smart Contract audit, Blockchain infrastructure Penetration testing and the Cryptocurrency forensic services with very affordable prices.

To Know More

Twitter: https://twitter.com/secureverse

LinkedIn: https://www.linkedin.com/company/secureverse/

Telegram: https://t.me/secureverse

Email Address: http://secureverse@protonmail.com/