Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: OKB Token
Website: okx.com
Platform: Ethereum
Language: Solidity
Date: March 5th, 2024

Table of contents

Email: audit@EtherAuthority.io

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.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contract audit initiatives, the OKB Token smart contract from okx.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 March 5th, 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

- This Solidity contract seems to be an implementation of a token contract for OKB (the cryptocurrency of OKEx exchange) with functionalities like transferring tokens, pausing/unpausing transfers, approving transfers, managing allowances, freezing/unfreezing addresses, and adjusting token supply.
- Here's a breakdown of the key components and functionalities:
    - **SafeMath Library:** This library is used for arithmetic operations to prevent overflows and underflows.
    - **Contract Variables:**
        - balances: Mapping to store the balance of each address.
        - **totalSupply:** Total supply of the token.
        - **name, symbol, decimals:** Metadata of the token.
        - **_allowed:** Mapping to track allowances for each address.
        - **owner:** Address of the contract owner.
        - **paused:** Boolean flag to control whether transfers are paused or not.
        - **lawEnforcementRole:** Address with special permissions.
        - **frozen:** Mapping to track frozen addresses.
        - **supplyController:** Address controlling token supply.
    - **Events:** Events are used to log significant contract actions.
    - **Modifiers:**
        - **onlyOwner:** Restricts functions to be called only by the contract owner.

        - **whenNotPaused:** Restricts functions to be called only when transfers are not paused.
        - **onlyLawEnforcementRole:** Restricts functions to be called only by the law enforcement role.
        - **onlySupplyController:** Restricts functions to be called only by the supply controller.
    - **Functions:**
        - **initialize():** Initializes the contract.
        - **totalSupply(), balanceOf(), allowance():** Getters for total supply, balance of an address, and allowance of spender.
        - **transfer(), transferFrom():** Transfers tokens from one address to another.
        - **approve(), increaseAllowance(), decreaseAllowance():** Approves spender to spend tokens on behalf of the owner.
        - **pause(), unpause():** Pauses/unpauses transfers.
        - **setLawEnforcementRole(), wipeFrozenAddress(), freeze(), unfreeze(), :** Manage frozen addresses by the law enforcement role.
        - **setSupplyController(), increaseSupply(), decreaseSupply():** Manage token supply.
- The OKBImplementation contract is a Pausable ERC20 token with Burn and Mint controlled by a central supply controller.
- This contract also includes external methods for setting a new implementation contract for the proxy.
- OKB Token smart contracts offer various functions like pause/unpause contracts and freeze/unfreeze address balances for transferring.
- Overall, this contract provides functionalities for managing a token ecosystem including ownership, transfers, allowances, freezing addresses, and controlling token supply, with certain permissions delegated to specific roles.

Audit scope

Name	Code Review and Security Analysis Report for OKB Token Smart Contract
Platform	**Ethereum**
File	OKBImplementation.sol
Smart Contract Code	0x5dba7dfcdbfb8812d30fdd99d9441f8b7a605621
Audit Date	March 5th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: OKB • Symbol: OKB • Decimals: 18	**YES, This is valid.**
Ownership Control: • Allows the current owner to transfer control of the contract to a new owner. • Pause / Unpause contract. • Freezes or unfreezes an address balance, allowing transfer by the law enforcement role or owner. • Wipes frozen address can be set by the law enforcement role or owner. • Sets a new supply controller addressed by the supply controller or owner. • Increases or decreases the total supply by minting the specified number of tokens into the supply controller's account.	**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 smart contracts are **"Poor Secured"**. Also, this contract contains owner control, which does not make it 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. A general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 1 critical, 0 high, 0 medium, 1 low, and 9 very low level issues.

Investors Advice: 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 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	Moderated
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

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?	Yes
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Yes
Blacklist Check	No
Can Mint?	Yes
Is it Proxy?	Yes
Can Take Ownership?	Not detected
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract file. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in OKB 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 OKB 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 an OKB 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 are 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

OKb.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	Passed	No Issue
3	totalSupply	read	Passed	No Issue
4	transfer	write	Passed	No Issue
5	balanceOf	read	Gas Optimization	Refer Audit Findings
6	transferFrom	write	Gas estimation failed, Gas Optimization	Refer Audit Findings
7	approve	write	Gas Optimization	Refer Audit Findings
8	_approve	internal	Passed	No Issue
9	allowance	read	Gas Optimization	Refer Audit Findings
10	increaseAllowance	write	Gas estimation failed, Gas Optimization	Refer Audit Findings
11	decreaseAllowance	write	Gas estimation failed, Gas Optimization	Refer Audit Findings
12	onlyOwner	modifier	Passed	No Issue
13	transferOwnership	write	Gas Optimization, Centralization	Refer Audit Findings
14	whenNotPaused	modifier	Passed	No Issue
15	pause	write	Centralization	Refer Audit Findings
16	unpause	write	Centralization	Refer Audit Findings
17	setLawEnforcementRole	write	Gas Optimization	Refer Audit Findings
18	onlyLawEnforcementRole	modifier	Passed	No Issue
19	freeze	write	Freeze account, Centralized risk	Refer Audit Findings
20	unfreeze	write	Centralization	Refer Audit Findings
21	wipeFrozenAddress	write	Centralized Risk, Gas estimation failed, Gas Optimization, Centralized risk	Refer Audit Findings
22	isFrozen	read	Gas Optimization	Refer Audit Findings
23	setSupplyController	write	Gas Optimization	Refer Audit Findings
24	onlySupplyController	modifier	Passed	No Issue

25	increaseSupply	write	Gas estimation failed, Gas Optimization, Unlimited token minting, Centralization	Refer Audit Findings
26	decreaseSupply	write	Gas estimation failed, Gas Optimization, Centralization	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 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 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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

(1) Centralized risk: **OKB.sol**

```
function freeze(address _addr) public onlyLawEnforcementRole {
    require(!frozen[_addr], "address already frozen");
    frozen[_addr] = true;
    emit AddressFrozen(_addr);
}
```

The owner has special permission to freeze others' addresses to restrict interaction to the smart contract.

```
function wipeFrozenAddress(address _addr) public onlyLawEnforcementRole {
    require(frozen[_addr], "address is not frozen");
    uint256 _balance = balances[_addr];
    balances[_addr] = 0;
    totalSupply_ = totalSupply_.sub(_balance);
    emit FrozenAddressWiped(_addr);
    emit SupplyDecreased(_addr, _balance);
    emit Transfer(_addr, address(0), _balance);
}
```

The owner has special permission to burn the balance of others.

Resolution: We suggest making smart contracts 100% decentralized.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Unlimited token minting: **OKb.sol**

```
/**
 * @dev Increases the total supply by minting the specified number of tokens to the supply co
 * @param _value The number of tokens to add.
 * @return A boolean that indicates if the operation was successful.
 */
function increaseSupply(uint256 _value) public onlySupplyController returns (bool success) {
    totalSupply_ = totalSupply_.add(_value);
    balances[supplyController] = balances[supplyController].add(_value);
    emit SupplyIncreased(supplyController, _value);
    emit Transfer(address(0), supplyController, _value);
    return true;
}
```

Token minting without any maximum limit is considered inappropriate for tokenomics.

Resolution: We recommend placing some limit on token minting to mitigate this issue.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: **OKb.sol, SafeMath.sol**

```
3   pragma solidity ^0.4.24;
```

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

Resolution: Please use versions greater than 0.8.7.

(2) Solidity constants naming conventions:

OKb.sol

```
string public constant name = "OKB"; // solium-disable-line uppercase
string public constant symbol = "OKB"; // solium-disable-line uppercase
uint8 public constant decimals = 18; // solium-disable-line uppercase
```

Constants are defined in lowercase.

Resolution: Constants should be named with all capital letters with underscores separating words.

(3) Gas Optimization:

OKb.sol

The public functions that are never called by the contract could be declared external.

- approve
- decreaseAllowance
- decreaseSupply
- increaseAllowance
- increaseSupply
- setLawEnforcementRole
- setSupplyController
- transferFrom
- transferOwnership
- wipeFrozenAddress
- allowance
- balanceOf
- isFrozen

Resolution: We suggest declaring this function external for better Gas optimization.

(4) Freeze account: **OKb.sol**

```
function freeze(address _addr) public onlyLawEnforcementRole {
    require(!frozen[_addr], "address already frozen");
    frozen[_addr] = true;
    emit AddressFrozen(_addr);
}
```

The onlyLawEnforcementRole can Freeze the account of any user.

Resolution: We suggest making your smart contract 100% decentralized.

(5) Centralization: **OKb.sol**

In the contract onlyOwner as an owner authority on the following function:

- transferOwnership
- pause

- unpause

In the contractor onlySupplyController as an owner authority on the following function:

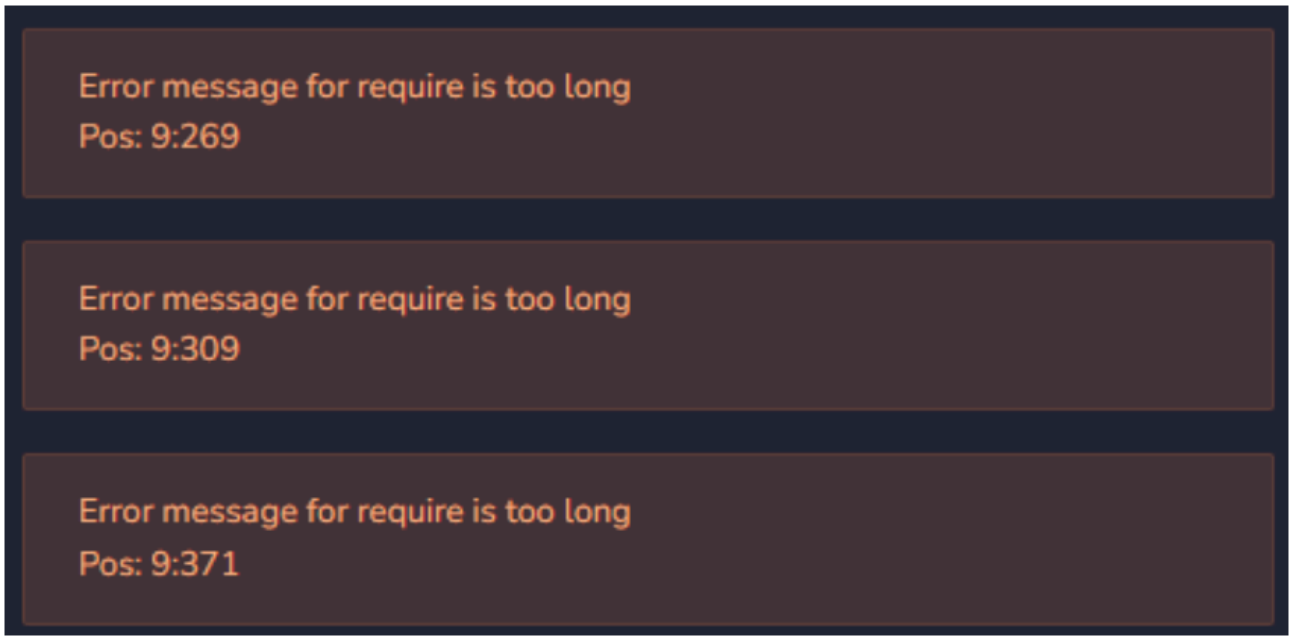
- increaseSupply
- decreaseSupply

In the contract onlyLawEnforcementRole as an owner authority on the following function:

- unfreeze
- wipeFrozenAddress

Resolution: Any compromise to these accounts may allow the hacker to manipulate the project through these functions We suggest carefully managing the owner account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

(6) Error message for require is too long: **OKb.sol**



Ethereum has a gas limit for each block. This limit includes the gas used by all transactions and contract executions within that block. When a required statement fails, it results in an exception, and the error message, along with the gas used up to that point, is included in the transaction's revert message.

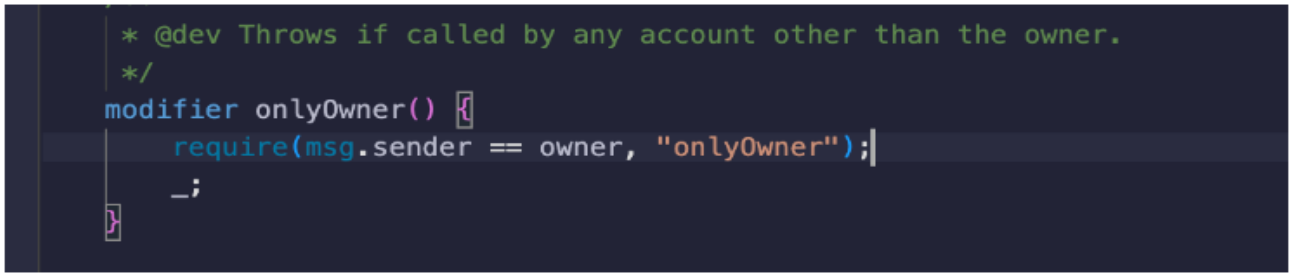
Resolution: We suggest writing short and clear messages in the required statement.

(7) Gas estimation failed: **OKb.sol**

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

Resolution: We suggest breaking them down into smaller functions or optimized code.

(8) Use openzeppelin's access control mechanism: **OKb.sol**



Written own logic to grant ownership mechanism to contract but it is a good practice to use openzeppelin ownable contract to implement access control mechanism.

Resolution: Use openzeppelin import "@openzeppelin/contracts/access/Ownable.sol";

(9) Use openzepplin's pausable library to pause and unpause token transfers, minting and burning: **OKB.sol**

```
279      */
280     modifier whenNotPaused() {
281         require(!paused, "whenNotPaused");
282         _;
283     }
284
285     /**
286      * @dev called by the owner to pause, triggers stopped state
287      */
288     function pause() public onlyOwner {
289         require(!paused, "already paused");
290         paused = true;
291         emit Pause();
292     }
293
294     /**
295      * @dev called by the owner to unpause, returns to normal state
296      */
297     function unpause() public onlyOwner {
298         require(paused, "already unpaused");
299         paused = false;
300         emit Unpause();
301     }
```

Written own logic to pause and unpause the contract but it's a good practice to use openzepplin's pausable library to pause and unpause token transfers, minting and burning.

Resolution: use openzeppelin link:
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. Following are Admin functions:

OKBImplementation.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.
- pause: Triggers stopped by the owner.
- unpause: Returns to normal state by the owner.
- setLawEnforcementRole: Sets a new law enforcement role address only lawEnforcementRole or Owner.
- freeze: Freezes an address balance from being transferred by the lawEnforcementRole or Owner.
- unfreeze: Unfreezes an address balance allowing transfer by the lawEnforcementRole or Owner.
- wipeFrozenAddress: Wipes frozen address can be set by the law enforcement role or Owner.
- setSupplyController: Sets a new supply controller addressed by the SupplyController or Owner.
- increaseSupply: Increases the total supply by minting the specified number of tokens to the supply controller account by the SupplyController.
- decreaseSupply: Decreases the total supply by burning the specified number of tokens from the supply controller account by the SupplyController.

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 1 critical, 1 low, and 9 informational issues in the smart contract. **So, the smart contract is ready for the mainnet deployment after resolving those issues.**

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 **"Poor Secured"**.

Email: audit@EtherAuthority.io

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 whitebox 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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

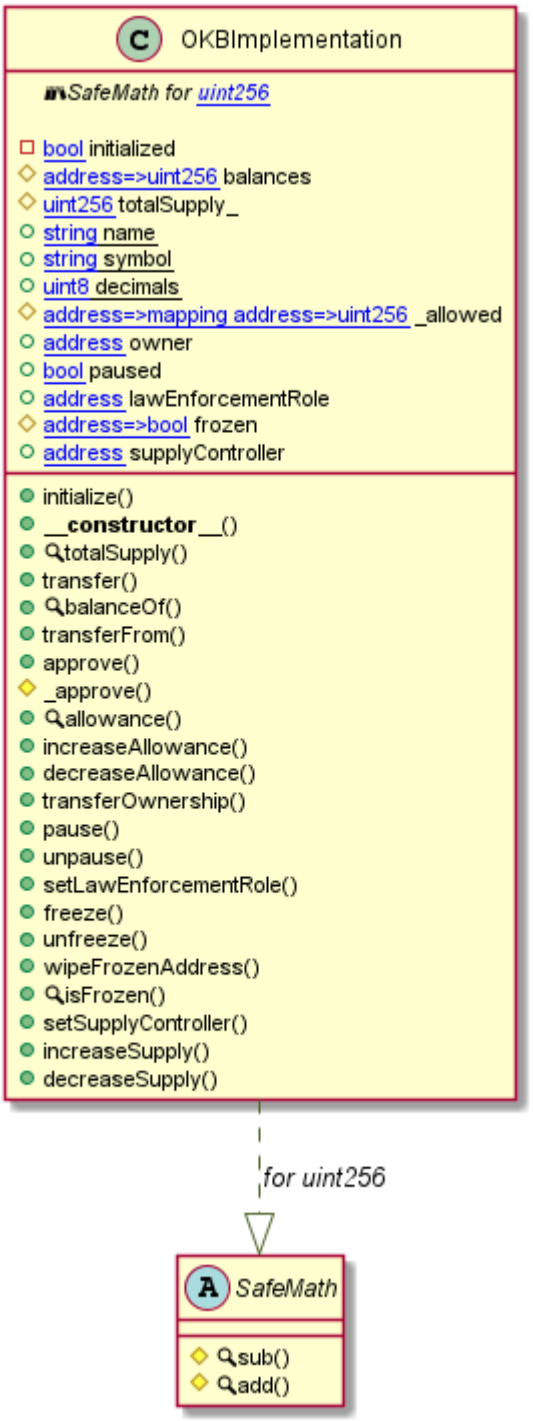
Technical Disclaimer

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 - OKB Token

OKBImplementation Diagram



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 >> OKBImplementation.sol

```
Pragma version^0.4.24 (OKBImplementation.sol#1) 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

Parameter OKBImplementation.transfer(address,uint256)._to (OKBImplementation.sol#170) is not in mixedCase
Parameter OKBImplementation.transfer(address,uint256)._value (OKBImplementation.sol#170) is not in mixedCase
Parameter OKBImplementation.balanceOf(address)._addr (OKBImplementation.sol#186) is not in mixedCase
Parameter OKBImplementation.transferFrom(address,address,uint256)._from (OKBImplementation.sol#198) is not in mixedCase
Parameter OKBImplementation.transferFrom(address,address,uint256)._to (OKBImplementation.sol#198) is not in mixedCase
Parameter OKBImplementation.transferFrom(address,address,uint256)._value (OKBImplementation.sol#198) is not in mixedCase
Parameter OKBImplementation.allowance(address,address)._owner (OKBImplementation.sol#244) is not in mixedCase
Parameter OKBImplementation.transferOwnership(address)._newOwner (OKBImplementation.sol#292) is not in mixedCase
Parameter OKBImplementation.setLawEnforcementRole(address)._newLawEnforcementRole (OKBImplementation.sol#332) is not in mixedC
ase
Parameter OKBImplementation.freeze(address)._addr (OKBImplementation.sol#348) is not in mixedCase
Parameter OKBImplementation.unfreeze(address)._addr (OKBImplementation.sol#358) is not in mixedCase
Parameter OKBImplementation.wipeFrozenAddress(address)._addr (OKBImplementation.sol#368) is not in mixedCase
Parameter OKBImplementation.isFrozen(address)._addr (OKBImplementation.sol#383) is not in mixedCase
Parameter OKBImplementation.setSupplyController(address)._newSupplyController (OKBImplementation.sol#393) is not in mixedCase
Parameter OKBImplementation.increaseSupply(uint256)._value (OKBImplementation.sol#410) is not in mixedCase
Parameter OKBImplementation.decreaseSupply(uint256)._value (OKBImplementation.sol#423) is not in mixedCase
Variable OKBImplementation._allowed (OKBImplementation.sol#66) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
OKBImplementation.sol analyzed (2 contracts with 84 detectors), 19 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.

OKBImplementation.sol

Gas costs:

Gas requirement of function OKBImplementation.decreaseSupply 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: 423:4:

Similar variable names:

OKBImplementation.wipeFrozenAddress(address) : Variables have very similar names "balances" and "_balance". Note: Modifiers are currently not considered by this static analysis.
Pos: 370:8:

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: 401:8:

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: 424:8:

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.

OKBImplementation.sol

```
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:0
Provide an error message for require
Pos: 9:12
Provide an error message for require
Pos: 9:23
Constant name must be in capitalized SNAKE_CASE
Pos: 5:60
Constant name must be in capitalized SNAKE_CASE
Pos: 5:61
Constant name must be in capitalized SNAKE_CASE
Pos: 5:62
Error message for require is too long
Pos: 9:292
Error message for require is too long
Pos: 9:332
Error message for require is too long
Pos: 9:394
```

Software analysis result:

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

Email: audit@EtherAuthority.io