QuillAudits

Audit Report
September, 2022

For

Key Calm Exchange

Table of Content

Table of Content

Executive Summary

Project Name	KEYCALM
Overview	KEYCALM is an ERC20 crowdsale contract utilizing in-line libraries for IERC20, Context and SafeMath. It has a total supply of 12,000,000 tokens with no mint or burn functionality for adjustment of supply later on.
Timeline	September 8th, 2022 to September 23th, 2022.
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze KEYCALM smart contract codebase for quality, security, and correctness. *https://github.com/Kilbury/crowdsale/blob/main/keycalm.sol* *SmartContractAudit/* *commit/001944ca04ab7860cc6534bbad283911fdbdc321* **Commit hash:** e09238da6961e11804d7a16cc25a46989e35f08b
Fixed In	*https://github.com/Kilbury/crowdsale/blob/main/keycalm.sol* **Commit hash:** f187fa340b72e73f52edf318e10e2b224dddb6f5

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	2
Partially Resolved Issues	0	0	1	0
Resolved Issues	0	0	1	5

Types of Severities

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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

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

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

High Severity Issues

No issues found

Medium Severity Issues

A.1 No access control implemented

Description

There is no access control implemented in the contract and as such, key functions (approve, transfer, transferFrom) are vulnerable to malicious actors. Any user can approve any amount of tokens to be spent and transfer to themselves. MarketResearch, DesigningAndDevelopment, InvestorsUsersAcquisition, UpgradeAndMaintenance, AdministrativeLegalExpenses, and Others addresses can be drained totally of funds.

Remediation

Implement proper access control, and a multi-sig wallet to reduce centralization risk as well.

Status

Acknowledged

Low Severity Issues

A.2 Hardcoded addresses are not advisable

Description

These addresses are hardcoded into the contract: MarketResearch, DesigningAndDevelopment, InvestorsUsersAcquisition, UpgradeAndMaintenance, AdministrativeLegalExpenses, and Others. There is also no setter function for any of these addresses. Funds in these hard-coded wallets are at greater risk of compromise.

Remediation

Set these addresses at contract initialization and have a setter function to update them later if needed. If preferred as it is, non-changing variables should be set to “constant” to save gas.

Status

Partially Resolved

A.3 Static balances declared

Description

Balances of MarketResearch, DesigningAndDevelopment, InvestorsUsersAcquisition, UpgradeAndMaintenance, AdministrativeLegalExpenses, and Others are hardcoded in the contract and do not change even when transfers are made out of them.

Remediation

Make the balances dynamically return the value in the wallet addresses specified, using the balanceOf function in IERC20 implementation.

Status

Resolved

Informational Issues

A.4 State mutability of decimals can be made pure

Description

The decimals function does not cause modifications to state. It is advisable to stick to the principle of least privilege granted.

```
function decimals() public view returns (uint8) {
    return 18;
}
```

Remediation

Change visibility modifier to pure, it also saves gas

Status

Resolved

A.5 Usage of outdated libraries and abstract contracts

Description

There are more up-to-date releases for the external contracts and libraries used (SafeMath and Context).

Remediation

Consider adding a minimum amount to receive greater than zero, The minimum amount to receive may vary according to The token amount passed in while swapping for ETH.

Status

Acknowledged

A.6 Multiple mappings and variable declared but never used

Description

The mappings _allowed, _addressLocked, _finalSoldAmount, reEntrance are declared but never used, same with tokenPrice and deploymentTime

```
uint256 private tokenPrice;
uint256 private deploymentTime;
mapping(address => mapping(address => uint256)) private _allowed;
mapping(address => bool) _addressLocked;
mapping(address => uint256) _finalSoldAmount;
mapping(address => mapping(uint256 => bool)) reEntrance;
```

Remediation

Implement these mappings or remove them from the contract.

Status

Resolved

A.7 Function declared but never used

Description

The internal function _mint is declared but not used in any other functions.

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");
    _beforeTokenTransfer(address(0), account, amount);
    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

Remediation

Unless mint functionality is to be implemented in contract at a later time, the function is unnecessary and can be removed to save gas

Status

Resolved

A.8 Arithmetic logic implemented in multiple ways

Description

The SafeMath library implemented for overflow and underflow checks is unnecessary because solidity versions above 0.8.0 have inbuilt overflow and underflow checks. Also, in setting amounts (MarketResearchAmount, DesigningAndDevelopmentAmount etc) normal multiplication is used instead of safemath's .mul

```
// without SafeMath
   uint256 public OthersAmount = 1200000 * (10**uint256(18));
// with SafeMath
   _balances[recipient] = _balances[recipient].add(amount);
```

Remediation

Consider removing Safemath's library completely or implement it everywhere an arithmetic function is required.

Status

Acknowledged

A.9 Constructor does not need visibility specifier

Description

Solidity versions >0.5.0 do not require a visibility specifier for constructors.

```
constructor() public {
```

Remediation

Removing public visibility specifier to prevent the compiler from throwing an error on compilation.

Status

Resolved

A.10 Unnecessary casting to uint256

Description

The multiplication to get various amounts with decimals inclusive has a uint256 cast that may be unnecessary, and could be declared with exponential (e) or the ether suffix for better readability.

```
uint256 public OthersAmount = 1200000 * (10**uint256(18));
```

Remediation

Consider removing the uint256 cast and using a numbering system that would be less prone to error.

Status

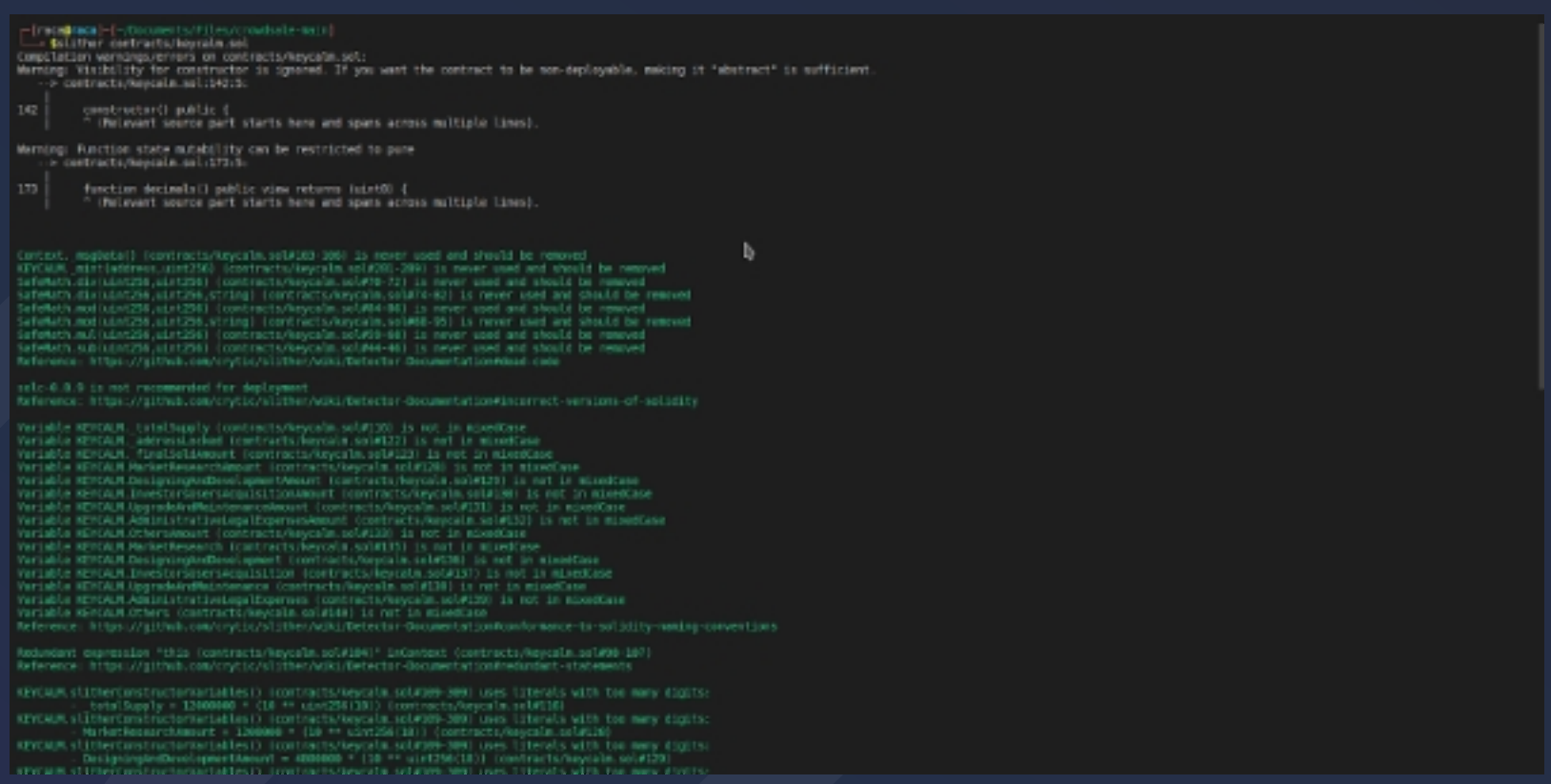
Resolved

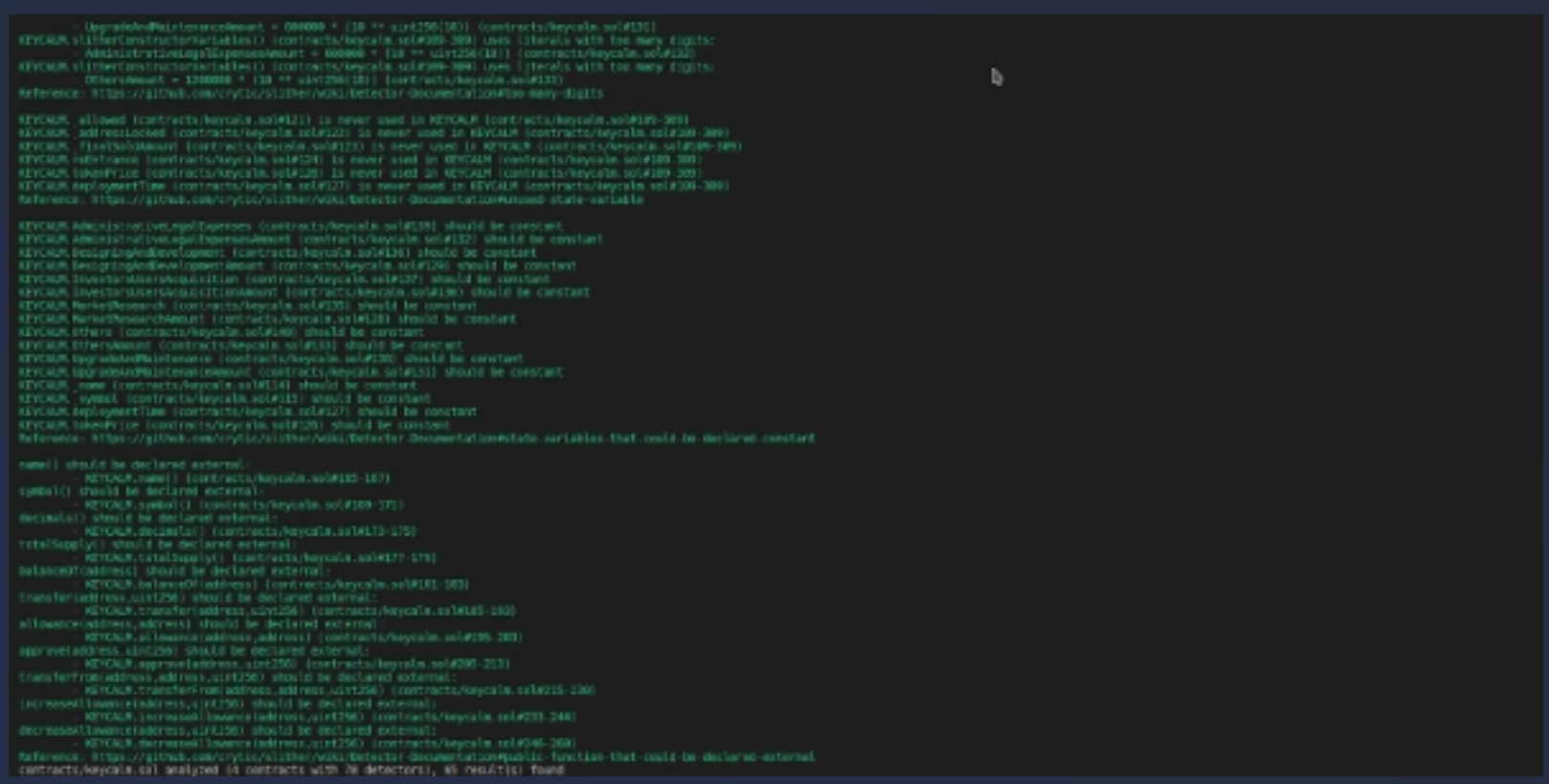
Functional Testing

- Should return the name, symbol, decimals, totalSupply
- Should transfer only the allowed amounts
- Should increase allowance by stated amount
- Should not decrease allowances below 0
- Should fail on transfers below given allowance
- Should return the actual balance of hardcoded addresses after transfers

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.





Important Note to User

Transfers from this crwdsale contract have been described to happen with an API and not a batch/whitelist process as normally occurs. Be aware that such API calls or requests are not included in the scope of this audit.

Closing Summary

In this report, we have considered the security of the KEYCALM contract. We performed our audit according to the procedure described above.

Some issues of Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the KEYCALM Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the KEYCALM Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com