KAI PROTOCOL
SECURITY ASSESSMENT REPORT

MAY. 4 - MAY. 18, 2021
MAY. 26 - JUN. 4, 2021
JUN. 13 - JUN. 14, 2021

DISCLAIMER

- This document is based on a security assessment conducted by a blockchain security company SOOHO. This document describes the detected security vulnerabilities and also discusses the code quality and code license violations.
- This security assessment does not guarantee nor describe the usefulness of the code, the stability of the code, the suitability of the business model, the legal regulation of the business, the suitability of the contract, and the bug-free status. Audit document is used for discussion purposes only.
- SOOHO does not disclose any business information obtained during the review or save it through a separate media.
- SOOHO presents its best endeavors in smart contract security assessment.

SOOHO

SOOHO with the motto of “Audit Everything, Automatically” researches and provides technology for reliable blockchain ecosystem. SOOHO verifies vulnerabilities through entire development life-cycle with Aegis, a vulnerability analyzer created by SOOHO, and open source analyzers. SOOHO is composed of experts including Ph.D researchers in the field of automated security tools and white-hackers verifying contract codes and detected vulnerabilities in depth. Professional experts in SOOHO secure partners’ contracts from known to zero-day vulnerabilities.

INTRODUCTION

SOOHO conducted a security assessment of KAI Protocol's smart contract from May 4 to May 18, 2021, May 26 to June 4, 2021 and Jun 13 to Jun 14, 2021. The following tasks were performed during the audit period:

- Performing and analyzing the results of Odin, a static analyzer of SOOHO.
- Writing Exploit codes on suspected vulnerability in the contract.
- Recommendations on codes based on best practices and the Secure Coding Guide.

A total of three security experts participated in a vulnerability analysis of the Cube System contract. The experts are professional hackers with Ph.D. academic backgrounds and experiences of receiving awards from national/international hacking competitions such as Defcon, Nuit du Hack, White Hat, SamsungCTF, and etc.

We scanned about known vulnerable codes through SOOHO's Odin in contracts. We have also conducted a more diverse security vulnerability detecting process with useful security tools.

The detected vulnerabilities are as follows: Critical 3, High 3, Low 3 and Note 1. It is recommended to promote the stability of service through continuous code audit and analyze potential vulnerabilities.

ANALYSIS TARGET

The following projects were analyzed from May 4 to May 18, 2021, May 26 to June 4, 2021 and Jun 13 to Jun 14, 2021.

Project	kdollar-contract-20210428	Project	kdollar-contract-20210525	Project	kaiprotocol-core-20210614
File Hash	f6fd2135	File Hash	ba3f5f03	File Hash	e1e7d92f
# of Files	35	# of Files	46	# of Files	27
# of Lines	2,622	# of Lines	3,404	# of Lines	2,069

KEY AUDIT POINTS & PROCESS

KAI Protocol is a algorithmic stable coin projects. KAI Protocol is originated from Basis Cash and Basis Dollar project. Shares and Bonds are used for stablize the dollar price. Accordingly, we mainly reviewed common vulnerabilities in token and possible hacking scenarios.

For example, the following scenarios are included: access control, input validation, token vesting logics, parameter validation. However, we did not take any internal hackings by administrators into account (e.g., Rug Pull). In addition, the design of protocol is also excluded from technical review.

1

Automated Vulnerability Analysis

2

Manual Code Analysis

The followings are considered:

- Preferential analysis of codes with greater risks.
- Supervision of Access Control management.
- Analyze whether the code is written under the client's intention.

3

Review of Exploitability and PoC Code

The followings are considered:

- Dynamic analysis through code execution.
- Examine possible financial gain by misusing detected vulnerabilities. (e.g., infinite withdrawal)
- Examine the possibility of adverse effect to the token service by misusing detected vulnerabilities. (e.g., Mint)

RISK RATING OF VULNERABILITY

Detected vulnerabilities are listed on the basis of the risk rating of vulnerability.

Critical High Medium Low Note

The risk rating of vulnerability is set based on OWASP's Impact & Likelihood Risk Rating Methodology as seen on the right. Some issues were rated vulnerable aside from the corresponding model and the reasons are explained in the following results.

	Likelihood		
Impact	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Note	Low	Medium
	Severity		

SOOHO

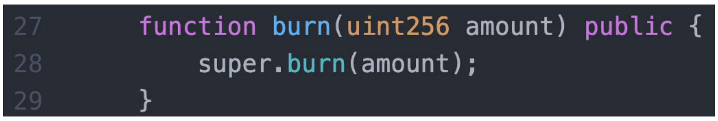
1st ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

(RESOLVED) ACCESS CONTROL High

File Name : `Bond.sol`

File Location : `kdollar-contract-20210428/assets/`
`└── Bond.sol`



```
    function burn(uint256 amount) public {
        super.burn(amount);
    }
```

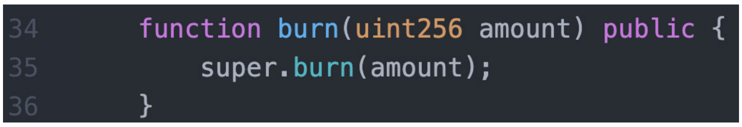
Details Since there is no access control in the `burn` function, the user can use it to burn the token arbitrarily. We advise the team to add the `onlyOperator` modifier for access control.

Additional resources and comments

(RESOLVED) ACCESS CONTROL High

File Name : `Cash.sol`

File Location : `kdollar-contract-20210428/assets/`
`└── Cash.sol`



```
    function burn(uint256 amount) public {
        super.burn(amount);
    }
```

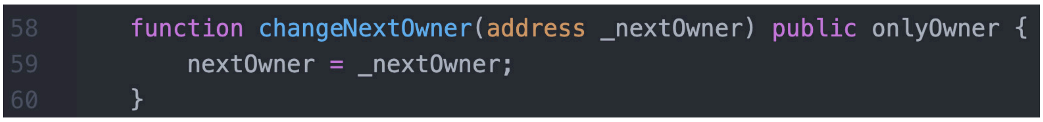
Details Since there is no access control in the `burn` function, the user can use it to burn the token arbitrarily. We advise the team to add the `onlyOperator` modifier for access control.

Additional resources and comments

(RESOLVED) NEXT OWNER CAN BE NULL High

File Name : `AirdropOperator.sol`

File Location : `kdollar-contract-20210428/`
`└── AirdropOperator.sol`



```
    function changeNextOwner(address _nextOwner) public onlyOwner {
        nextOwner = _nextOwner;
    }
```

Details When the `nextOwner` is assigned, the address can be Null since the validation is missing. If the `nextOwner` changed into Null, then the `changeOwner` will not work. It is recommended to add a Null check.

Additional resources and comments

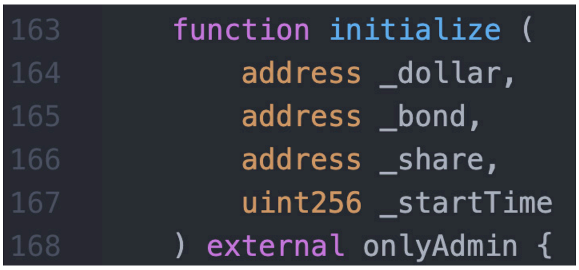
2nd ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

(RESOLVED) TREASURYIMPL CAN BE REINITIALIZED Critical

File Name : `TreasuryImpl.sol`

File Location : `kdollar-contract-20210525/`
`└── TreasuryImpl.sol`



```
    function initialize (
        address _dollar,
        address _bond,
        address _share,
        uint256 _startTime
    ) external onlyAdmin {
```

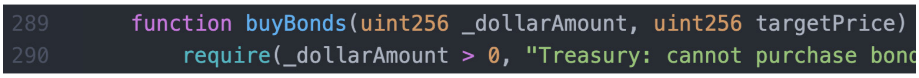
Details `initialize` can be executed multiple times. It should be call exactly once.

Additional resources and comments

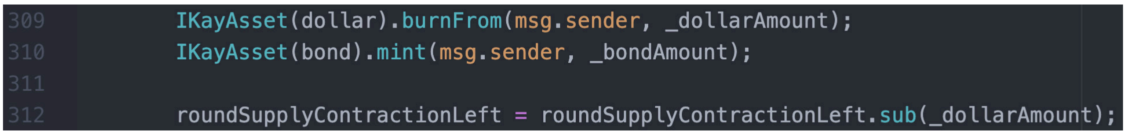
(NON ISSUE) DOLLAR PRICE SHOULD BE UPDATED Critical

File Name : `TreasuryImpl.sol`

File Location : `kdollar-contract-20210525/`
`└── TreasuryImpl.sol`

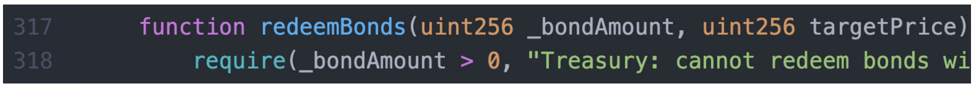


```
    function buyBonds(uint256 _dollarAmount, uint256 targetPrice)
        require(_dollarAmount > 0, "Treasury: cannot purchase bond
```

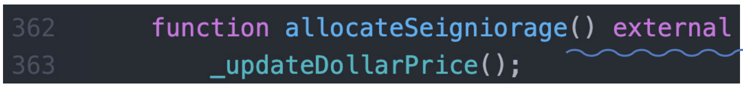


```
        IKayAsset(dollar).burnFrom(msg.sender, _dollarAmount);
        IKayAsset(bond).mint(msg.sender, _bondAmount);

        roundSupplyContractionLeft = roundSupplyContractionLeft.sub(_dollarAmount);
```



```
    function redeemBonds(uint256 _bondAmount, uint256 targetPrice)
        require(_bondAmount > 0, "Treasury: cannot redeem bonds wi
```



```
    function allocateSeigniorage() external
        _updateDollarPrice();
```

Details After burn the dollar and mint the bond in the function `buyBonds`, `_updateDollarPrice` function need to be executed. Similarly, `_updateDollarPrice` function should be called at the end of the function calls for `redeemBonds` and `allocateSeigniorage`, which affect the amount of bonds and the amount of dollars.

Additional resources and comments

The dev team confirmed that the update function is not needed since the price only changed by the swap. Thus, we changed it to non-issues.

2nd ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

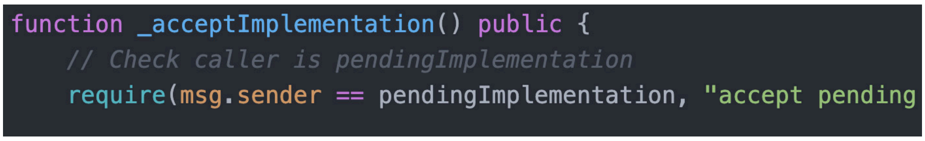
PENDINGIMPLEMENTATION CAN BE NULL Low

Additional resources and comments

File Name : `TreasuryUni.sol`

File Location : `kdollar-contract-20210525`

```
└── TreasuryUni.sol
```



```
function _acceptImplementation() public {
    // Check caller is pendingImplementation
    require(msg.sender == pendingImplementation, "accept pending
```

Details `pendingImplementation` can be null and thus `implementation` can also be null. We recommend to add `pendingImplementation != address(0)` similar to Compound project.

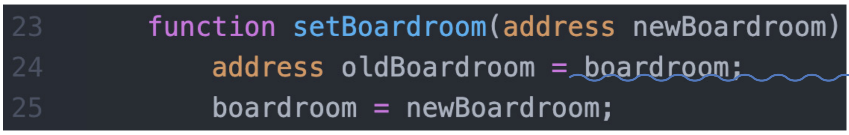
(RESOLVED) BOARD ROOM CAN BE NULL Low

Additional resources and comments

File Name : `VoteProxy.sol`

File Location : `kdollar-contract-20210525`

```
└── VoteProxy.sol
```



```
    function setBoardroom(address newBoardroom)
        address oldBoardroom = boardroom;
        boardroom = newBoardroom;
```

The file is excluded in the latest project.

Details We recommend checking whether the boardroom is null or not.

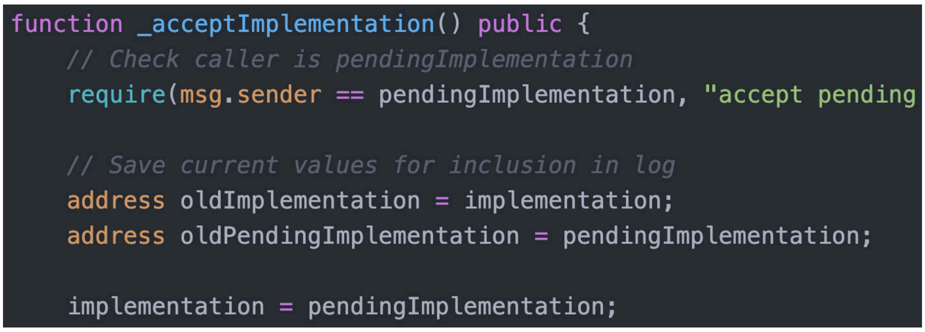
TREASURYIMPL CAN BE CHANGED Note

Additional resources and comments

File Name : `TreasuryUni.sol`

File Location : `kdollar-contract-20210525`

```
└── TreasuryUni.sol
```



```
function _acceptImplementation() public {
    // Check caller is pendingImplementation
    require(msg.sender == pendingImplementation, "accept pending

    // Save current values for inclusion in log
    address oldImplementation = implementation;
    address oldPendingImplementation = pendingImplementation;

    implementation = pendingImplementation;
```

Details The administrator can change the contract address where the logic for treasury is defined. It can be a potential threat since the logic can be changed without the users' consensus.

3rd ANALYSIS RESULTS

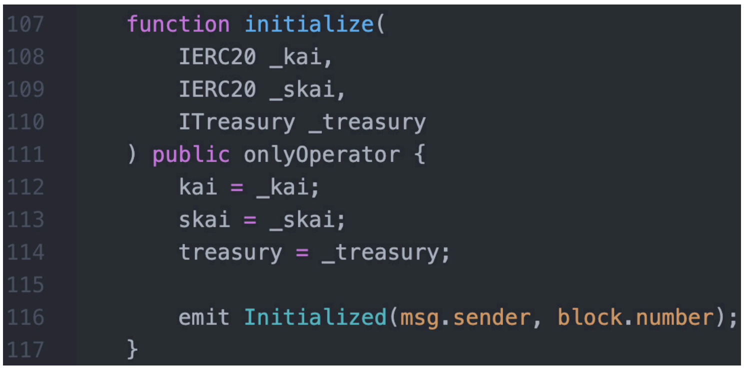
Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

BOARDROOM CAN BE REINITIALIZED Critical

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `Boardroom.sol`

File Location : `kaiprotocol-core-20210614`
```
└── Boardroom.sol
```



```
    function initialize(
        IERC20 _kai,
        IERC20 _skai,
        ITreasury _treasury
    ) public onlyOperator {
        kai = _kai;
        skai = _skai;
        treasury = _treasury;

        emit Initialized(msg.sender, block.number);
    }
```

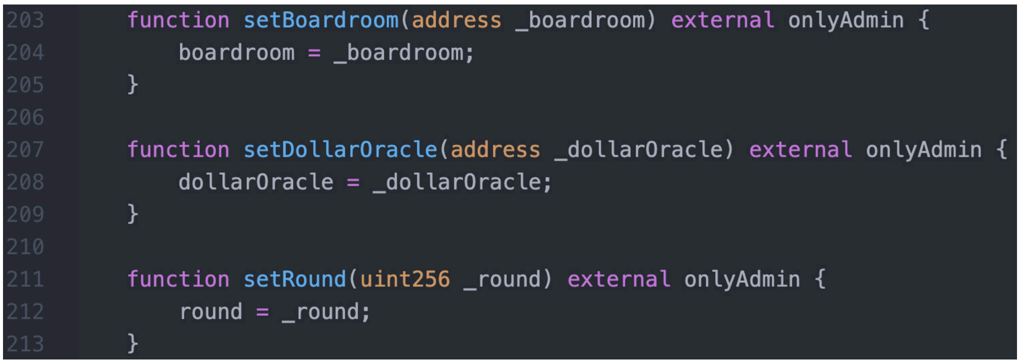
Details `initialize` should be call exactly once.

ADDRESS CAN BE NULL Low

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryImpl.sol`

File Location : `kaiprotocol-core-20210614`
```
└── TreasuryImpl.sol
```



```
    function setBoardroom(address _boardroom) external onlyAdmin {
        boardroom = _boardroom;
    }

    function setDollarOracle(address _dollarOracle) external onlyAdmin {
        dollarOracle = _dollarOracle;
    }

    function setRound(uint256 _round) external onlyAdmin {
        round = _round;
    }
```

Details `boardroom, dollarOracle, round` can be null. We recommend to check the null value.

ANALYSIS RESULTS

Additional analysis results include key issues that are not vulnerable but have been highlighted in the vulnerability analysis process.

ANALYZED - MATHEMATICAL OPERATIONS ✓

Details We have confirmed that the mathematical operations are working well.

Additional resources and comments

ANALYZED - REENTRANCY ✓

Details We analyzed possible reentrancy attacks in the contracts.

Additional resources and comments

CONCLUSIONS

The source code of the KAI Protocol is easy to read and very well organized. We have to remark that contracts were well handling the possible situations and writing test codes. Most of the codes are found out to be compliant with all the best practices. **The detected vulnerabilities are as follows: Critical 3, High 3, Low 3 and Note 1.** It is recommended to promote the stability of service through continuous code audit and analyze potential vulnerabilities.

Project	kaiprotocol-core-20210614
File Hash	e1e7d92f
# of Files	27
# of Lines	2,069

```
kaiprotocol-core-20210614
├── BBFund.sol
├── BBFundImpl.sol
├── BBFundStorage.sol
├── Boardroom.sol
├── OracleKlayswap.sol
├── Timelock.sol
├── TreasuryImpl.sol
├── TreasuryStorage.sol
├── TreasuryUni.sol
├── assets
│   ├── KAI.sol
│   ├── KAIBond.sol
│   ├── KAIShare.sol
│   └── kERC20.sol
├── interfaces
│   ├── IBoardroom.sol
│   ├── IKAIAsset.sol
│   ├── IKlayExchange.sol
│   ├── IKlayswapFactory.sol
│   ├── IKlayswapStore.sol
│   ├── IOracle.sol
│   └── ITreasury.sol
├── lib
│   ├── Babylonian.sol
│   ├── FixedPoint.sol
│   └── UQ112x112.sol
├── owner
│   ├── Operator.sol
│   └── Ownable.sol
└── utils
    ├── ContractGuard.sol
    └── Epoch.sol
```