BEOSIN
Blockchain Security

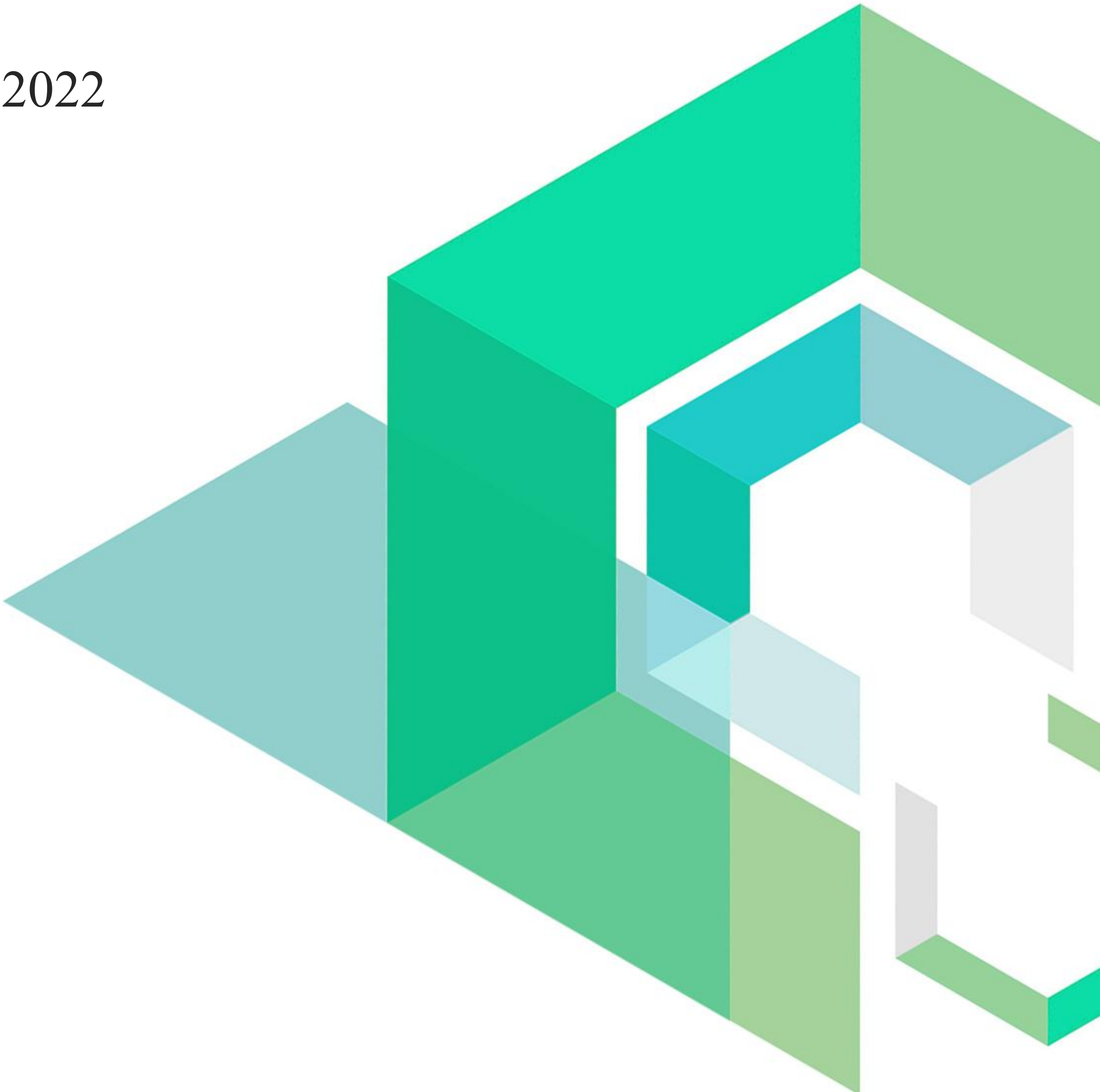
TME

Smart Contract Security Audit

V1.0

No. 202210251233

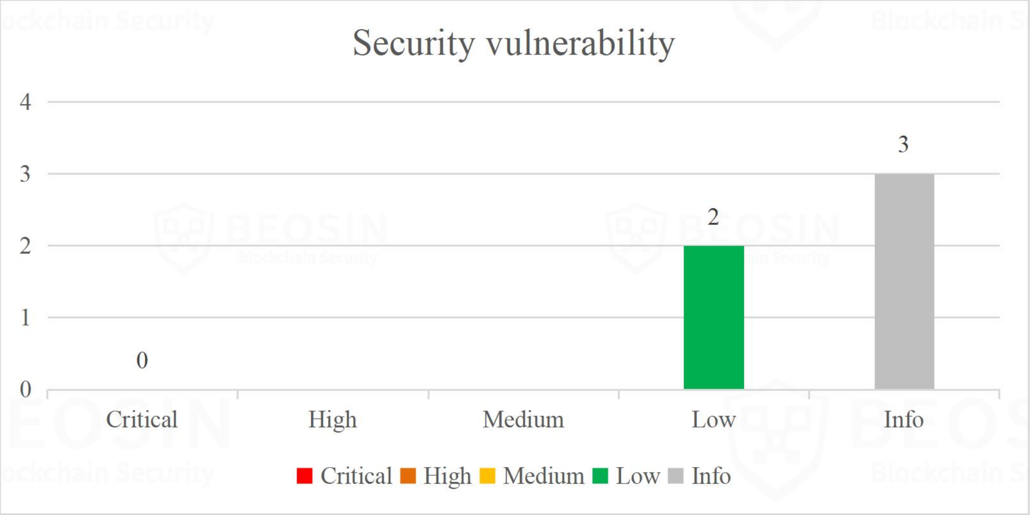
Oct 25th, 2022



Contents

Summary of Audit Results

After auditing, 2 Low-risk and 3 Info-risk items were identified in the TME project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Risk Description:**

1. This contract designs reward mechanism that rewards should be given during each transfer operation. However, the reward will not be applied because of reward factor equal to zero. User should pay attention that they will not receive the reward in the current contract of BSC mainnet.

2. There is asset centralization risk that this project mint all token to one address during contract depoly.

3. flpReceiver is external address designated by owner that used to obtain LP token through add liquidity operation. lpReceiver can withdraw USDT and TME token by remove liquidity through pair address.

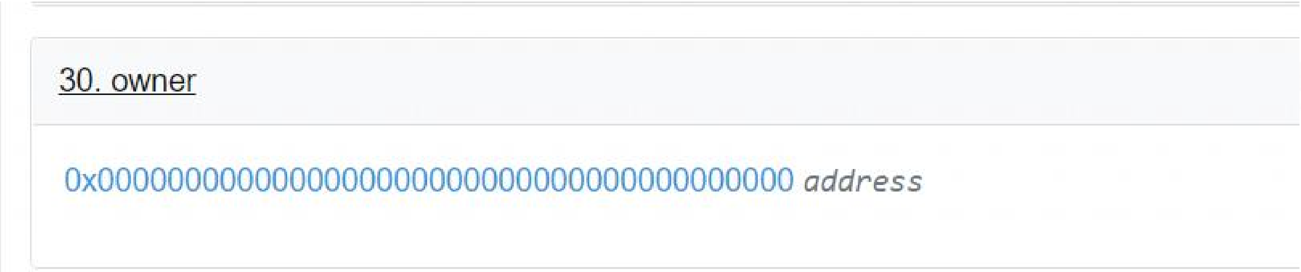
4. Small amount of USDT in the token contract that cannot be withdrawn.

- **Project Description:**

1. Business overview

The TME is a BEP-20 token issued on BNB Chain. The total supply of TME is 21 million, which can not be minted and can be burned (transfer to the dead address). The contract will mint the total supply of tokens to the deployer address when the contract is deployed. The deployer will be granted owner permission when the contract is deployed. This contract has reward mechanism. However, the reward will not be applied because of reward factor equal to zero. During each transfer operation, the contract takes a percentage of the fees and transfers them to a different addresses, including the current contract address. When the balance of the current contract exceeds a certain threshold, the add liquidity operation will be triggered.

The owner has the right to set important variables such as whitelist, fee rate, canTransfer, canSwap,etc. The owner also has the right to change the address of router, nftPool, fund, lpReceiver, etc. However, based on BNB chain data, the owner address had renounced ownership. Fee ratio is fixed and no longer to change because of owner renounce ownership. Based on BNB chain data, the fee ratio is follows: fundFeePercent 2.7%, marketFeePercent 0.5%, devFeePercent 0.3%, deadFeePercent 3%, liquidityFeePercent 3%, nftPoolFeePercent 0.5%.



30. owner

0x0000000000000000000000000000000000000000 *address*

2. Basic Token Information

Token name	The Micro Elements
Token symbol	TME
Decimals	18
Pre-mint	21,000,000 (All to deployer)
Total supply	21,000,000 (Burnable)
Token type	BEP-20

Table 1 Basic information of TME

1 Overview

1.1 Project Overview

Project Name	TME
Platform	BNB Chain
Contract Address	0x52b9f5ccdb313CA1125D8bf9Be800f78CeA15351

1.2 Audit Overview

Audit work duration: October 25, 2022 – October 25, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team

2 Findings

Index	Risk description	Severity level	Status
TME-1	Centralization risk	Low	Acknowledged
TME-2	The remaining USDT cannot be withdrawn	Low	Acknowledged
TME-3	Unreasonable setting of key parameters	Info	Acknowledged
TME-4	Redundant code	Info	Acknowledged
TME-5	The _totalSupply was not updated when the token was destroyed	Info	Acknowledged

Status Notes:

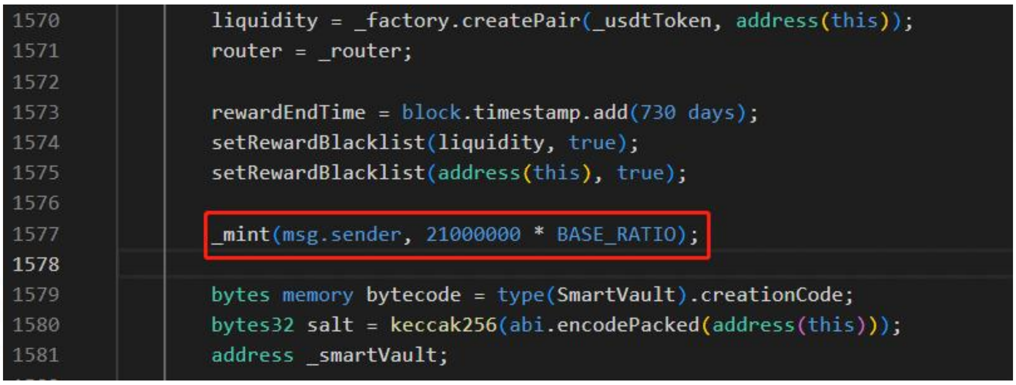
- TME-1 is unfixed and asset is centralized by one address.
- TME-2 is unfixed and small amount of USDT is locked in the contract.
- TME-3 is unfixed and users are not rewarded during transfer.
- TME-4 is unfixed and will not cause any issues.
- TME-5 is unfixed and will not cause any issues.

[TME-1] Centralization risk

Severity Level	**Low**
Type	Business Security
Lines	TMEToken.sol #L1577

Description

When the contract is deployed, all tokens are allocated to the deployer's account through the *_mint* function, which has the risk of centralization of token allocation.



```
1570        liquidity = _factory.createPair(_usdtToken, address(this));
1571        router = _router;
1572
1573        rewardEndTime = block.timestamp.add(730 days);
1574        setRewardBlacklist(liquidity, true);
1575        setRewardBlacklist(address(this), true);
1576
1577        _mint(msg.sender, 21000000 * BASE_RATIO);
1578
1579        bytes memory bytecode = type(SmartVault).creationCode;
1580        bytes32 salt = keccak256(abi.encodePacked(address(this)));
1581        address _smartVault;
```

Figure 1 Source code of *constructor* function

Recommendations

It is recommended to use multi-signature wallet, DAO or TimeLock to manage the pre-mint token.

Status

Acknowledged.

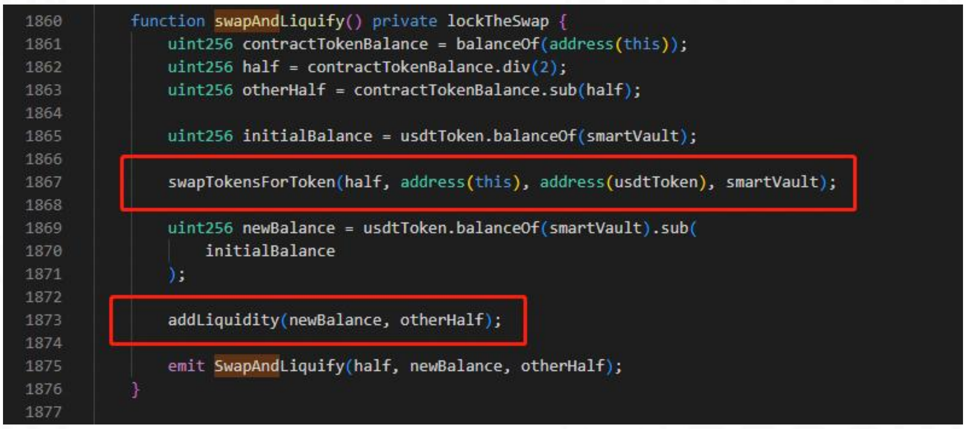
[TME-2] The remaining USDT cannot be withdrawn

Severity Level Low

Type Business Security

Lines TMEToken.sol #L1853-1868

Description The *swapAndLiquidy* function converts half of the contractTokenBalance of TME tokens to USDT. The other half of TME tokens and the converted USDT are deposited into the TME-USDT pool on PancakeSwap as liquidity. For every *swapAndLiquify* function call, a small amount of USDT leftover in the contract. This is because the price of TME drops after swapping the first half of TME tokens into USDT, and the other half of TME tokens require less than the converted USDT to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those USDT, and they will be locked in the contract permanent.



```
    function swapAndLiquify() private lockTheSwap {
        uint256 contractTokenBalance = balanceOf(address(this));
        uint256 half = contractTokenBalance.div(2);
        uint256 otherHalf = contractTokenBalance.sub(half);

        uint256 initialBalance = usdtToken.balanceOf(smartVault);

        swapTokensForToken(half, address(this), address(usdtToken), smartVault);

        uint256 newBalance = usdtToken.balanceOf(smartVault).sub(
            initialBalance
        );

        addLiquidity(newBalance, otherHalf);

        emit SwapAndLiquify(half, newBalance, otherHalf);
    }
```

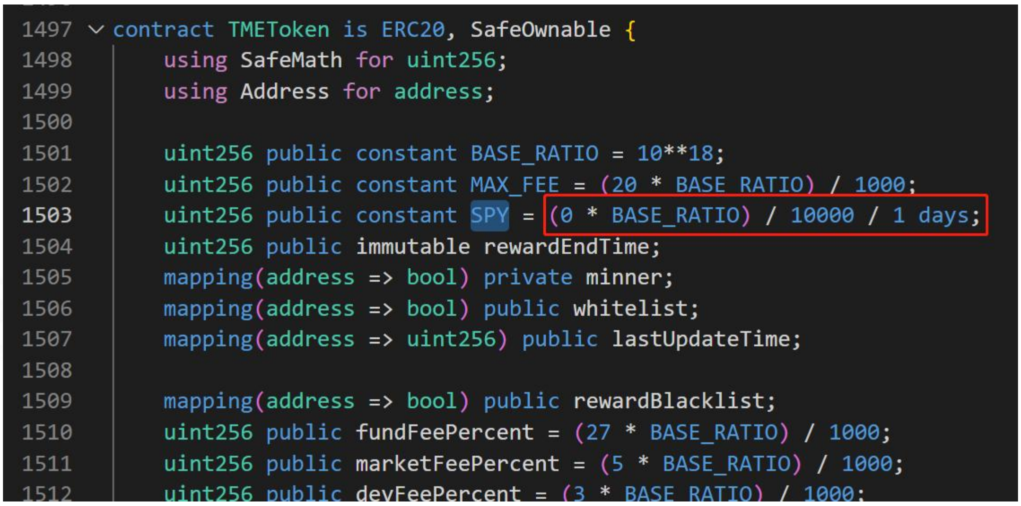
Figure 2 The source code of *swapAndLiquify* function

Recommendations It is recommended to add the function of drawing USDT in the contract.

Status Acknowledged.

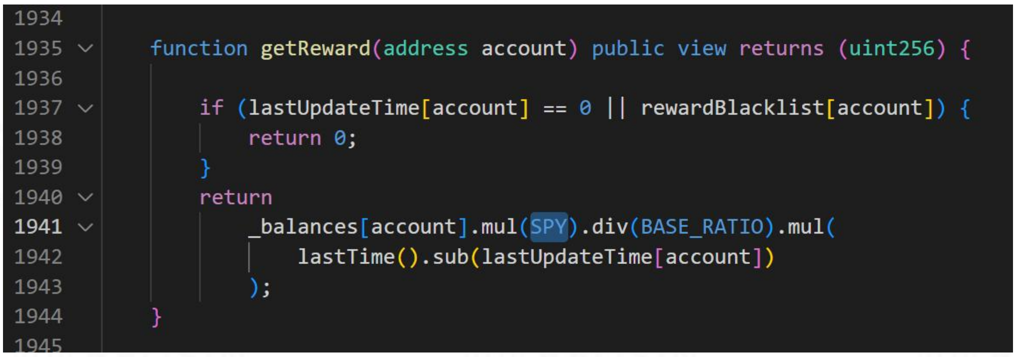
[TME-3] Unreasonable setting of key parameters

Severity Level	Info
Type	Business Security
Lines	TMEToken.sol #L1503
Description	The constant SPY equal to zero and can’t be change, which will cause the reward can’t be applied in transfer operation.



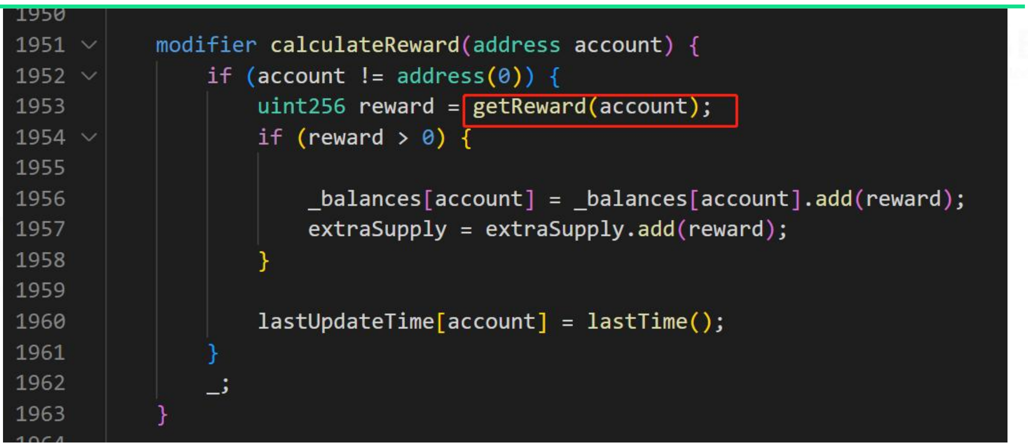
```
1497  contract TMEToken is ERC20, SafeOwnable {
1498      using SafeMath for uint256;
1499      using Address for address;
1500
1501      uint256 public constant BASE_RATIO = 10**18;
1502      uint256 public constant MAX_FEE = (20 * BASE_RATIO) / 1000;
1503      uint256 public constant SPY = (0 * BASE_RATIO) / 10000 / 1 days;
1504      uint256 public immutable rewardEndTime;
1505      mapping(address => bool) private minner;
1506      mapping(address => bool) public whitelist;
1507      mapping(address => uint256) public lastUpdateTime;
1508
1509      mapping(address => bool) public rewardBlacklist;
1510      uint256 public fundFeePercent = (27 * BASE_RATIO) / 1000;
1511      uint256 public marketFeePercent = (5 * BASE_RATIO) / 1000;
1512      uint256 public devFeePercent = (3 * BASE_RATIO) / 1000;
```

Figure 3 Source code of TME related code



```
1934
1935      function getReward(address account) public view returns (uint256) {
1936
1937          if (lastUpdateTime[account] == 0 || rewardBlacklist[account]) {
1938              return 0;
1939          }
1940          return
1941              _balances[account].mul(SPY).div(BASE_RATIO).mul(
1942                  lastTime().sub(lastUpdateTime[account])
1943              );
1944      }
1945
```

Figure 4 Source code of *getReward* function



```
    modifier calculateReward(address account) {
        if (account != address(0)) {
            uint256 reward = getReward(account);
            if (reward > 0) {

                _balances[account] = _balances[account].add(reward);
                extraSupply = extraSupply.add(reward);
            }

            lastUpdateTime[account] = lastTime();
        }
        _;
    }
```

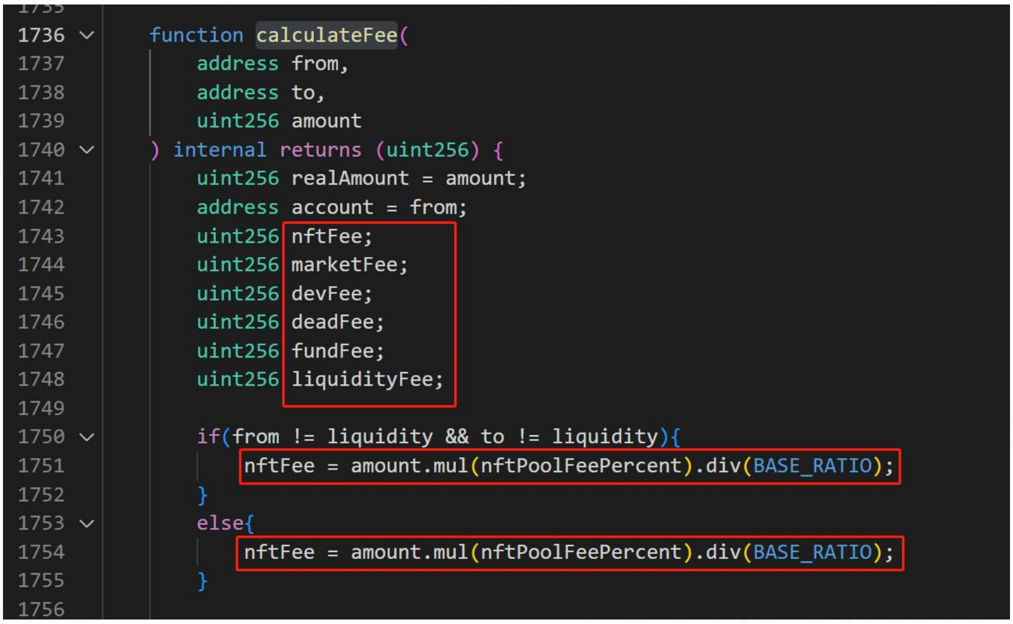
Figure 5 Source code of *calculateReward* modifier

Additionally, the destroyed tokens (sent to the dead address) will still participate in the reward calculation.

Recommendations	It is recommended to add function to set SPY value .
Status	Acknowledged. The project team is aware of the problem. However, the project was already running on BSC mainnet and SPY variable can't be modified.

[TME-4] Redundant code

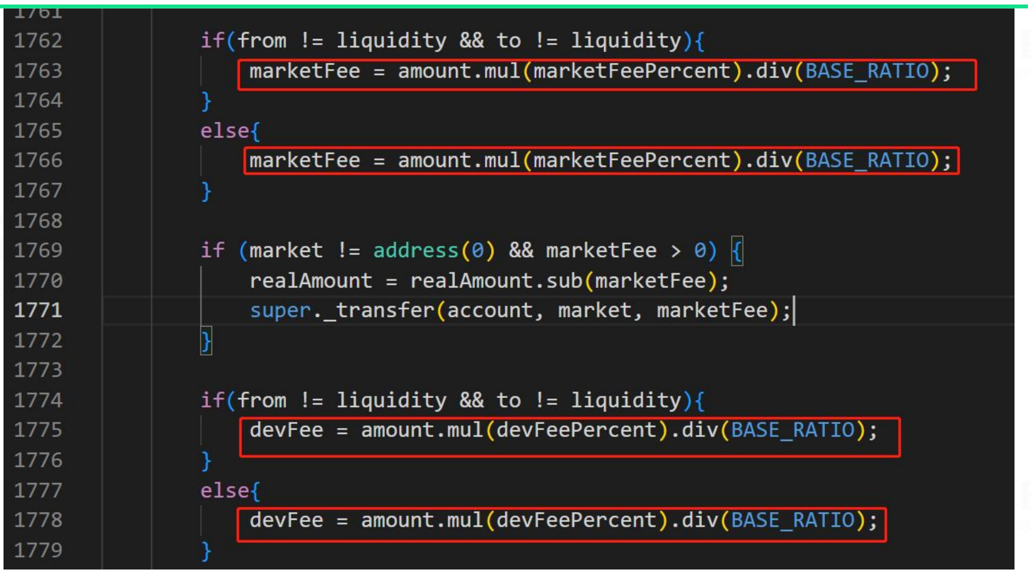
Severity Level	Info
Type	Coding Conventions
Lines	TMEToken.sol #L1736-1851 TMEToken.sol #L251 TMEToken.sol #L1522 TMEToken.sol #L1505
Description	Some conditions in *CalculateFee* function is useless. Furthermore some interfaces and variables are not used, such as IDayOfRightsClub, referralHandle, minner.



```
function calculateFee(
    address from,
    address to,
    uint256 amount
) internal returns (uint256) {
    uint256 realAmount = amount;
    address account = from;
    uint256 nftFee;
    uint256 marketFee;
    uint256 devFee;
    uint256 deadFee;
    uint256 fundFee;
    uint256 liquidityFee;

    if(from != liquidity && to != liquidity){
        nftFee = amount.mul(nftPoolFeePercent).div(BASE_RATIO);
    }
    else{
        nftFee = amount.mul(nftPoolFeePercent).div(BASE_RATIO);
    }
```

Figure 6 Source code of *calculateFee* function

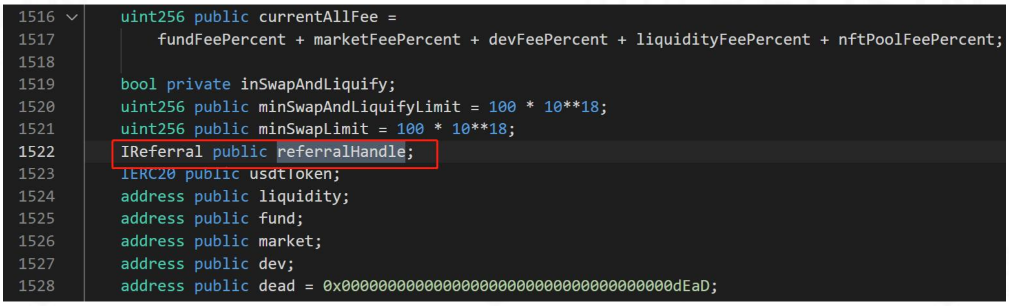


```
        if(from != liquidity && to != liquidity){
            marketFee = amount.mul(marketFeePercent).div(BASE_RATIO);
        }
        else{
            marketFee = amount.mul(marketFeePercent).div(BASE_RATIO);
        }

        if (market != address(0) && marketFee > 0) {
            realAmount = realAmount.sub(marketFee);
            super._transfer(account, market, marketFee);
        }

        if(from != liquidity && to != liquidity){
            devFee = amount.mul(devFeePercent).div(BASE_RATIO);
        }
        else{
            devFee = amount.mul(devFeePercent).div(BASE_RATIO);
        }
```

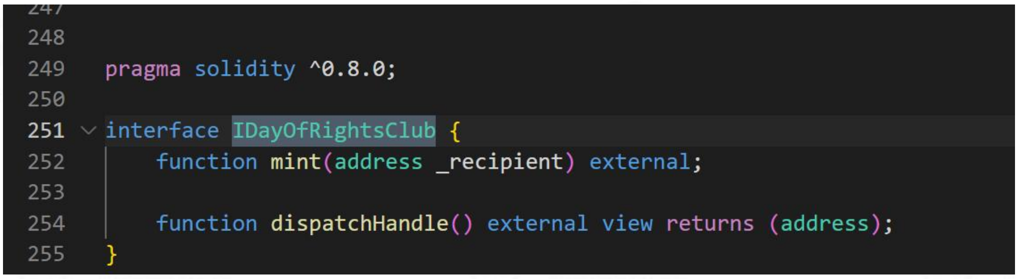
Figure 7 Source code of *calculateFee* function



```
    uint256 public currentAllFee =
        fundFeePercent + marketFeePercent + devFeePercent + liquidityFeePercent + nftPoolFeePercent;

    bool private inSwapAndLiquify;
    uint256 public minSwapAndLiquifyLimit = 100 * 10**18;
    uint256 public minSwapLimit = 100 * 10**18;
    IReferral public referralHandle;
    IERC20 public usdtToken;
    address public liquidity;
    address public fund;
    address public market;
    address public dev;
    address public dead = 0x000000000000000000000000000000000000dEaD;
```

Figure 8 Source code of TME related code

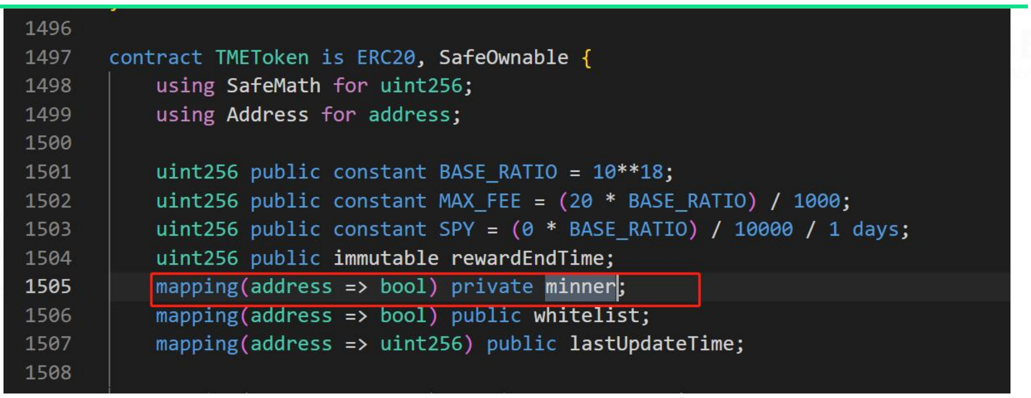


```
pragma solidity ^0.8.0;

interface IDayOfRightsClub {
    function mint(address _recipient) external;

    function dispatchHandle() external view returns (address);
}
```

Figure 9 Source code of IDayOfRightsClub interface



```
contract TMEToken is ERC20, SafeOwnable {
    using SafeMath for uint256;
    using Address for address;

    uint256 public constant BASE_RATIO = 10**18;
    uint256 public constant MAX_FEE = (20 * BASE_RATIO) / 1000;
    uint256 public constant SPY = (0 * BASE_RATIO) / 10000 / 1 days;
    uint256 public immutable rewardEndTime;
    mapping(address => bool) private minner;
    mapping(address => bool) public whitelist;
    mapping(address => uint256) public lastUpdateTime;
```

Figure 10 Source code of TME related code

Recommendations	It is recommended to delete redundant code.
Status	Acknowledged.

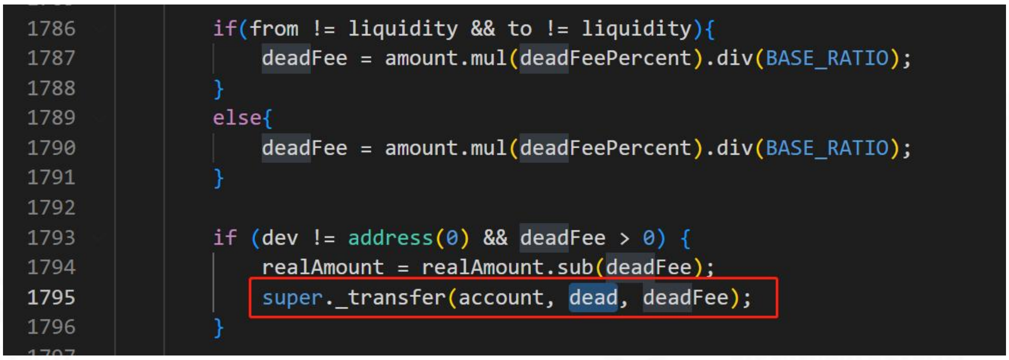
[TME-5] The _totalSupply was not updated when the token was destroyed

Severity Level Info

Type Business Security

Lines TMEToken.sol #L1795-1803

Description The token transferred to 0xdEaD for destruction is not recorded. It cause the displayed total supply to be inconsistent with the actual.



```
1786            if(from != liquidity && to != liquidity){
1787                deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1788            }
1789            else{
1790                deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1791            }
1792
1793            if (dev != address(0) && deadFee > 0) {
1794                realAmount = realAmount.sub(deadFee);
1795                super._transfer(account, dead, deadFee);
1796            }
```

Figure 11 The source code of related code

Recommendations It is recommended to add logic to update _totalSupply when destroying tokens.

Status Acknowledged.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides invesTME nt advice on any project, nor should it be utilized as invesTME nt suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com