SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

TOKEN MANAGER

Overview

Project Summary

- Name: Token Manager
- Platform: BSC Chain
- Language: Solidity
- Repository:
    - https://github.com/dev-four-d4/four-meme-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Token Manager
Version	v1
Type	Solidity
Dates	Mar 31 2025
Logs	Mar 31 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	3
Total Low-Severity issues	1
Total informational issues	2
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	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.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. addLiquidity failed to prevent token state change 6

2. sellToken function may be susceptible to a phishing attack 8

3. Slippage control may fail in certain situations 9

4. Centralization risk 10

5. The creator who cannot accept ETH will prevent addLiquidity 11

2.3 Informational Findings 12

6. Redundant code 12

7. Missing sanity check for pcfee 13

Appendix **14**

Appendix 1 - Files in Scope 14

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	addLiquidity failed to prevent token state change	High	Business Logic	Pending
2	sellToken function may be susceptible to a phishing attack	Medium	Business Logic	Pending
3	Slippage control may fail in certain situations	Medium	Business Logic	Pending
4	Centralization risk	Medium	Centralization	Pending
5	The creator who cannot accept ETH will prevent addLiquidity	Low	Business Logic	Pending
6	Redundant code	Informational	Gas optimization	Pending
7	Missing sanity check for pcfee	Informational	Business Logic	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. addLiquidity failed to prevent token state change

Severity: High	Category: Business Logic

Target:
- LibTM2.2.bsc.sol

Description

Due to price manipulation protection, when liquidity is added by `addLiquidity` and the target pair has already been created with a non-zero reserve, the operation will revert and wait for manual extraction. However, after manual extraction, the token's status cannot be changed. The token's mode will remain in `MODE_TRANSFER_RESTRICTED`, and the owner role will not be relinquished. This will result in transfer restrictions persisting during trading.

LibTM2.2.bsc.sol:L92-L105

```
function _isManualDeployRequired(address tokenAddr, address quoteAddr) internal view
returns (bool) {
    address pairAddr = IPancakeFactory(PANCAKE_FACTORY).getPair(tokenAddr, quoteAddr);
    if (pairAddr == address(0)) {
        return false;
    }

    (uint112 reserve0, uint112 reserve1,) = IPancakePair(pairAddr).getReserves();
    return (reserve0 > 0 || reserve1 > 0);
}

function _addLiquidityV2(address tokenAddr, address quoteAddr, uint256 tokenAmt, uint256
quoteAmt) internal {
    if (_isManualDeployRequired(tokenAddr, quoteAddr)) {
        revert("Revert");
    } else {
        ...
```

TokenManager3.6.bsc.sol:L486-L502

```
    function addLiquidity(address tokenAddress) public onlyOperator {
        TokenInfo storage ti = _tokenInfo(tokenAddress);
        require(ti.status == STATUS_ADDING_LIQUIDITY, "T");

        IToken token = IToken(tokenAddress);
        token.setMode(0);
        ...
        LibTM.addLiquidity(
            LibTM.dexOf(tix1.pcFee),
            LibTM.feeOf(tix1.pcFee),
            tokenAddress,
            ti.quote == address(0) ? LibTM.WETH : ti.quote,
            amountTokenDesired,
            amountFundsDesired);
        ...
    }
```

Recommendation

It is recommended to add relevant functions to allow the owner to adjust the mode of tokens that have manually added liquidity to `MODE_NORMAL` and relinquish the ownership or when `_isManualDeployRequired` returns true, do not revert the transaction, but instead trigger an event.

2. sellToken function may be susceptible to a phishing attack

Severity: Medium	Category: Business Logic
Target: - TokenManager3.6.bsc.sol	

Description

Using `tx.origin` to check the token holder in the `sellToken` function could lead to users being vulnerable to phishing attacks. If a user is misled into interacting with a malicious contract, the malicious contract could cause the user to unintentionally sell their tokens.

TokenManager3.6.bsc.sol:L423-L424

```
function sellToken(uint256 origin, address token, address from, uint256 amount, uint256
minFunds, uint256 feeRate, address feeRecipient) public nonReentrant {
    require(tx.origin == from, "Not holder");
    ...
}
```

Recommendation

It is recommended to use `msg.sender` to identify the owner instead of `tx.origin`.

3. Slippage control may fail in certain situations

Severity: Medium	Category: Business Logic
Target: - TokenManager3.6.bsc.sol	

Description

In the `_buyToken` function, when the remaining available token amount is insufficient, slippage control can fail.

When calculating the amount of tokens to purchase based on the input funds and slippage parameter (`minAmounts`), if `minAmounts > ti.offers`, the slippage parameter will be adjusted to `ti.offers`, causing the user's input slippage parameter to become ineffective.

Similarly, when calculating the purchase amount based on the input `amount` and slippage parameter (`maxFunds`), if `amount > ti.offers`, the calculation will use `ti.offers` to determine the required input funds, causing the user's input slippage parameter to become ineffective.

TokenManager3.6.bsc.sol:L350-L393

```
function _buyToken(BuyTokenParams memory params) internal returns (uint256) {
    TokenInfo storage ti = _tokenInfoForTrade(params.token);

    if (params.amount > 0) {
        require(params.amount % GWEI == 0, "GW");
    } else if (params.funds > 0) {
        params.amount = _calcBuyTokenAMAP(ti, params.funds,
            params.minAmount > ti.offers ? ti.offers : params.minAmount);
    }
    ...

    if (params.amount > ti.offers) {
        params.amount = ti.offers;
    }

    uint256 funds = calcBuyCost(ti, params.amount);
    uint256 fee = calcTradingFee(ti, funds);
    require(params.maxFunds == 0 || funds <= params.maxFunds, "Price is too high");
    ...
}
```

Recommendation

It is recommended to allow users to input a parameter to control whether partial fulfillment is accepted when the available offers are insufficient. Otherwise, the user's submitted slippage parameters should not be modified.

4. Centralization risk

Severity: Medium	Category: Centralization
Target: - TokenManager3.6.bsc.sol	

Description

The protocol has a privileged role, the owner, who can withdraw ETH and ERC20 tokens from the protocol at will.

Should the owner's private key be compromised, an attacker could withdraw all funds in the protocol.

contracts/oracle/Oracle.sol: L676-L682

```
function withdrawEth(address to, uint256 amount) public onlyOwner {
    Address.sendValue(payable(to), amount);
}

function withdrawERC20(address token, address to, uint256 amount) public onlyOwner {
    SafeERC20.safeTransfer(IERC20(token), to, amount);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

5. The creator who cannot accept ETH will prevent addLiquidity

Severity: Low	Category: Business Logic
Target: - TokenManager3.6.bsc.sol	

Description

In the `addLiquidity` function, there is a step involving the transfer of ETH to the creator. If the creator is unable to accept ETH, the entire transaction will revert, preventing the addition of liquidity.

TokenManager3.6.bsc.sol:L423-L424

```
function addLiquidity(address tokenAddress) public onlyOperator {
    ...
    if (tix.creator != address(0)) {
        uint256 raisingFee = ti.funds / 100;
        if (ti.quote == address(0)) {
            Address.sendValue(payable(_feeRecipient), raisingFee);
            Address.sendValue(payable(tix.creator), raisingFee);
        } else {
        ...
    }
    ...
}
```

Recommendation

It is recommended that when a transfer fails, the transaction should not revert. Instead, the unclaimed funds should be stored in a mapping, allowing the creator to claim them later.

2.3 Informational Findings

6. Redundant code

Severity: Informational	Category: Gas optimization
Target: - TokenManager3.6.bsc.sol	

Description

The contract contains some code related to deprecated logic.

TokenManager3.6.bsc.sol:L486-L502

```
function setFeeCollector(address account, bool enabled) public onlyOwner {
    _feeCollectors[account] = enabled;
}

function collectFees(uint256 lockId, address recipient, uint128 amount0Max, uint128
amount1Max)
    external nonReentrant
    returns (uint256 amount0, uint256 amount1, uint256 fee0, uint256 fee1)
{
    require(_feeCollectors[_msgSender()], "N");

    (amount0, amount1, fee0, fee1) = IUniV3LPLocker(LibTM.LP_LOCKER).collect(
        lockId,
        recipient,
        amount0Max,
        amount1Max
    );

    emit FeesCollected(lockId);
}
```

TokenManager3.6.bsc.sol:L716-L721

```
function lockLP(uint256[] memory nftIds) public onlyOperator {
    for (uint256 i = 0; i < nftIds.length; i++) {
        uint256 lockId = LibTM.lockLP(0, nftIds[i], address(this), _feeRecipient);
        emit LPLock(address(0), nftIds[i], lockId);
    }
}
```

Recommendation

It is recommended to remove this section of code to improve readability and reduce deployment costs.

7. Missing sanity check for pcfee

Severity: Informational	Category: Business Logic
Target: - TokenManager3.6.bsc.sol	

Description

The `pcfee` parameter contains the fee and DEX information used when adding liquidity. However, the current contract only allows liquidity addition through the `DEX_PANCAKE` type. Therefore, if the `pcfee` is unchecked, it could create tokens that are unable to addLiquidity.

TokenManager3.6.bsc.sol:L612-L618

```
function createToken(bytes memory args, bytes memory signature) public payable {
    ...
    _tokenInfoEx1s[token] = TokenInfoEx1({
        launchFee: a.payLaunchFeeByTrader && a.origin == 0 ? _launchFee : 0,
        pcFee: a.pcFee,
        reserved2: 0,
        reserved3: 0,
        reserved4: 0
    });
    ...
}
```

Recommendation

It is recommended to check if the dex information in the `pcfee` parameter is `DEX_PANCAKE`.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit a1d0ea9:

File	SHA-1 hash
TokenManager3.6.bsc.sol	ffc314c6be339a4d30ece1424fa430ee24f5818a
LibTM2.2.bsc.sol	39003f69c9c6a8e8dde237e6e4d764b618887bc7
TokenCreator2.bsc.sol	598d5f4a87cd5d67bca9d776e9b2d1f7d78911b9
Token2.sol	054cb7cd9894322227af832a51b83b70293a07ea