SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

FOUR MEME

Overview

Project Summary

- Name: Four Meme
- Platform: Binance Smart Chain
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Four Meme
Version	v5
Type	Solidity
Dates	Oct 09 2024
Logs	Sep 07 2024, Sep 09 2024; Sep 14 2024, Oct 08 2024; Oct 09 2024

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	7
Total Low-Severity issues	0
Total informational issues	1
Total	11

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

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	Meme tokens may be locked in contract	High	Business Logic	Resolved
2	The meme pool launch can be delayed	High	Business Logic	Resolved
3	The initial price can be manipulated, causes meme to fail to flow to liquidity pools	High	Business Logic	Resolved
4	Non-functional LP Locking	Medium	Business Logic	Resolved
5	Centralization risk	Medium	Centralization	Acknowledged
6	Possible billing dos because of small supply	Medium	Business Logic	Resolved
7	The voteUserNum can be manipulated	Medium	Data Validation	Resolved
8	Chain reorg may lead to voting on the wrong token	Medium	Front-running	Resolved
9	Id in addToken may be reused	Medium	Data Validation	Resolved
10	Add token can be dos	Medium	Business Logic	Resolved
11	Lack of _disableInitializers	Informational	Business Logic	Resolved

2.2 Notable Findings

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

1. Meme tokens may be locked in contract

Severity: High	Category: Business Logic
Target: - contracts/Activity.sol	

Description

When the activity is finished, the manager can withdraw meme tokens in the contract.

The problem is that the available withdrawal meme token amount cannot exceed the VOTE_TOKEN's balance. And VOTE_TOKEN's balance might be zero because users can claim their VOTE_TOKEN before the activity is totally finished.

contracts/Activity.sol:L285-L293

```
function adminWithdraw(IERC20 token, uint amount) external onlyRole(MANAGER_ROLE) {
    require(feeReceiver != address(0), "illegal receiver");
    // Only the whole activity finished, we can withdraw meme
    require(block.timestamp >= timeInfo.finishTime() + OWNER_WITHDRAW_TIME, "illegal
time");
    uint balance = VOTE_TOKEN.balanceOf(address(this));
    if (amount > balance) {
        amount = balance;
    }

    token.safeTransfer(feeReceiver, amount);
}
```

Recommendation

Should check the withdrawal token's balance.

Status

This issue has been resolved by the team.

2. The meme Pool launch can be delayed.

Severity: High	Category: Business Logic
Target: - contracts/Activity.sol	

Description

After a meme token vote ends, the `KEEPER_ROLE` can call the `billing` function to launch the meme token. The `billing` function involves `addLiquidity` to the meme-baseToken pair through the `SWAP_ROUTER`.
A malicious user can create a meme-baseToken pair before the `KEEPER_ROLE` calls the `billing` function, then transfer 1 wei of `baseToken` to the pair and sync to update the baseToken reserve.
This way, calling the `billing` function to `addLiquidity` will revert because the check meme token reserve is equal to 0, while the `baseToken` reserve is greater than 0 in `SWAP_ROUTER::quote`.
contracts/Activity.sol:L202-L211

```
function billing(uint64 season) external payable onlyRole(KEEPER_ROLE)
whenBilling(season) {
    ...
    ( , , uint liquidity) = SWAP_ROUTER.addLiquidity(
        address(memeToken), address(BASE_TOKEN),
        memeTokenAmount, baseTokenAmount,
        0, 0,
        address(this), block.timestamp
    );
    ...
}
```

contracts/mock/swap/PancakeRouter.sol:L308-L312

```
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint
amountB) {
    require(amountA > 0, 'PancakeLibrary: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'PancakeLibrary: INSUFFICIENT_LIQUIDITY');
}
```

Recommendation

We recommend adding a new function that allows the `KEEPER_ROLE` to directly send meme tokens and `baseToken` to the meme-baseToken pair contract and call `mint` to add liquidity when such situations occur.

Status

This issue has been resolved by the team.

3. The initial price can be manipulated, causes meme to fail to flow to liquidity pools

Severity: High	Category: Business Logic
Target: - contracts/Activity.sol	

Description

The protocol uses PancakeV3 as its liquidity pool. When there is no liquidity in the pool, its price can be arbitrarily manipulated. This occurs because, when liquidity is insufficient or price limits are reached, the pool executes partial trades instead of reverting.

If the initial price is manipulated, it may result in meme and baseToken being injected into the liquidity pool at an incorrect ratio during the billing phase. Additionally, if the price manipulation is effective, it could prevent meme from entering the liquidity pool during the billing phase.

Attach Scenario

1. Before the keeper calls billing to add liquidity, attackers manipulate the price of solt0 in the pool to the highest or lowest
2. When adding tokens required for liquidity, only a single token is provided because the price is outside the set range.
3. Therefore, attackers can lock memeToken in the activity contract instead of flowing to the pair

Proof of Concept

This test demonstrates that after a malicious user manipulates the price through a swap, the expected behavior of billing changes (meme tokens fail to flow properly into the liquidity pool).

```
function test_billing_exploit() public {
    address memeToken = poolInit();

    vm.warp(block.timestamp + 1000);
    vm.startPrank(exploiter);
    IPancakeV3Pool pair = IPancakeV3Pool(
        pancakeFactory.getPool(address(baseToken), address(memeToken), 2500)
    );

    bool isZero = address(memeToken) < address(baseToken) ? false : true;

    Callback callback = new Callback(
        address(pair),
        address(baseToken),
        address(memeToken)
    );
    bytes memory call = new bytes(0);
```

```
    // Price manipulation
    callback.swap(
        msg.sender,
        isZero,
        1,
        isZero
            ? 4295128739 + 1
            : 1461446703485210103287273052203988822378723970342 - 1,
        call
    );
    vm.stopPrank();

    vm.startPrank(admin);
    uint256 memeBeforeBilling = IERC20(memeToken).balanceOf(address(activity));
    activity.billing(1);
    uint256 memeAfterBilling = IERC20(memeToken).balanceOf(address(activity));

    require(memeBeforeBilling == memeAfterBilling, "Meme is temporarily locked in the
activity contract");
}
```

Recommendation

Consider adding features to ensure meme's enter the liquidity pool at billing time.

Status

This issue has been resolved by the team.

4. Non-functional LP Locking

Severity: Medium	Category: Business Logic
Target: - contracts/Activity.sol	

Description

When the keeper bills one season, LP tokens will be locked for ten years.

The problem is that lockLPToken's parameter should be the unlock date, not locking period.

src/Activity.sol:L187-L231

```
function billing(uint64 season) external payable onlyRole(KEEPER_ROLE)
whenBilling(season) {
    ...
    IERC20(pair).approve(address(LP_LOCKER), liquidity);
    LP_LOCKER.lockLPToken{value: ethFee}(pair, liquidity, LOCK_PERIOD,
payable(address(0)), true, feeReceiver);
    ...
}
```

Recommendation

Should use `current.timestamp + LOCK_PERIOD`.

Status

This issue has been resolved by the team.

5. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/Activity.sol - contracts/Factory.sol	

Description

In Activity and Factory contracts, there exists one privileged owner role. This role has authority over key operations such as upgrading the contract.

If these roles' private keys were compromised, an attacker could exploit this access to withdraw all tokens.

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.

Status

This issue has been acknowledged by the team.

6. Possible billing dos because of small supply

Severity: Medium	Category: Business Logic
Target: - contracts/Activity.sol	

Description

When the keeper bills one season, some meme tokens will be added into LP pool as the liquidity.

The problem is that the meme token amount for LP pool might be too small if the total supply is quite small. There is one `MINIMUM_LIQUIDITY` limitation in the Uniswap V2 pool. The `addLiquidity` might be reverted because of insufficient liquidity.

src/Activity.sol:L188-L198

```
function billing(uint64 season) external payable onlyRole(KEEPER_ROLE)
whenBilling(season) {
    require(pairs[season] == address(0), "already billing");
    address bestTokenAddress = bestToken[season];
    ...
    // Get the best meme token from this season.
    IERC20 memeToken = IERC20(bestToken[season]);
    uint totalSupply = memeToken.totalSupply();
    uint memeTokenAmount = totalSupply * memeTokenPercent / PERCENT_BASE;
    ...
}
```

Recommendation

Add one minimum totalSupply limitation when users add one token.

Status

This issue has been resolved by the team.

7. The voteUserNum can be manipulated

Severity: Medium	Category: Data Validation
Target: - contracts/Activity.so	

Description

In the `vote` function, when `voteHistory[msg.sender][tokenAddress] == 0`, it will increase the `voteUserNum` for that token.

Since there is no check to verify whether the input `amount` is larger than 0, a malicious actor can repeatedly call the `vote` function with an `amount` of 0 to continuously increase the `voteUserNum`. The `voteUserNum` will be manipulated, affecting other users' voting choice.

contracts/Activity.sol:L165-L181

```
function vote(address tokenAddress, uint amount) external whenNotPaused whenVoting
nonReentrant {
    ...
    bool newUser = voteHistory[msg.sender][tokenAddress] == 0;
    if (newUser) {
        tokenInfo.voteUserNum += 1;
    }
    ...
    voteHistory[msg.sender][tokenAddress] = voteHistory[msg.sender][tokenAddress] +
amount;
    ...
}
```

Recommendation

Check that the `amount` must be greater than 0.

Status

This issue has been resolved by the team.

8. Chain reorg may lead to voting on the wrong token.

Severity: Medium	Category: Front-running
Target: - contracts/Activity.so - contracts/Factory.sol	

Description

The `Factory` contract uses the `create` opcode to deploy new ERC20 tokens, so the address of the deployed contract is determined by the `nonce` of the `Factory` contract.

If a chain reorganization occurs, it might reverse the addresses of two meme tokens deployed before it happened. Voters may vote for the wrong meme token.

Attach Scenario
The transaction order before a chain reorganization occurs:
1. Alice added a meme token, which was deployed at `addrA`.
2. Alice voted 1000 for `addrA`.

The transaction order after a chain reorganization occurs:
1. Bob added a meme token, which was deployed at `addrA`
2. Alice added a meme token, which was deployed at `addrB`.
3. Alice voted 1000 for `addrA`.
This chain reorganization causes Alice to mistakenly vote 1000 for the meme token created by Bob.

contracts/Factory.sol:L81-L83

```
function deployERC20(string calldata _name, string calldata _symbol, uint256
_totalSupply) external returns (FourMemeERC20) {
    return new FourMemeERC20(_name, _symbol, defaultDecimals, _totalSupply, msg.sender);
}
```

contracts/Activity.sol:L155-L185

```
function vote(address tokenAddress, uint amount) external whenNotPaused whenVoting
nonReentrant {
    ...
}
```

Recommendation

We recommend using the `create2` method, specifying the hash of the creator's address and the token information as the salt.

Status

This issue has been resolved by the team.

9. Id in addToken function may be reused

Severity: Medium	Category: Data Validation
Target: - contracts/Activity.sol	

Description

The `addToken` function does not perform any checks on the provided id, name, symbol, or other parameters. This means a malicious actor could add a token with information identical to that of an already deployed token. In the current design, the `id` is uniquely generated by the backend and it should not be reused. Deploying two tokens with identical information should also not be encouraged.

contracts/Activity.sol:L131-L153

```
    function addToken(bytes32 id, string calldata name, string calldata symbol, uint
totalSupply) external payable whenNotPaused whenVoting nonReentrant {
        ...
        IERC20 token = factory.deployERC20(name, symbol, totalSupply);
        ...
        emit AddToken(id, token, name, symbol, factory.defaultDecimals(), totalSupply,
season, block.timestamp);
    }
```

Recommendation

We recommend generating the id based on the token information on-chain to distinguish between different tokens.

Status

This issue has been resolved by the team.

10. Add token can be dos

Severity: Medium	Category: Business Logic
Target: - contracts/Activity.sol	

Description

contracts/Activity.sol:L259-L304

```
function addToken(bytes32 id, string calldata name, string calldata symbol, uint
totalSupply) external payable whenNotPaused whenVoting nonReentrant {
    ...
    IERC20 token = factory.deployERC20(tokenId, name, symbol, totalSupply);
    createV3Pool(token);
    ...
}

function createV3Pool(IERC20 memeToken) internal {
    ...
    address pool = SWAP_V3_FACTORY.createPool(token0, token1, SWAP_V3_FEE);
    IPancakeV3Pool(pool).initialize(sqrtPriceX96);
    tokenPairs[address(memeToken)] = pool;
}
```

The `addToken()` function needs to call the createPool() in pancakeV3 Factory.

PancakeV3Factory

```
/// @inheritdoc IPancakeV3Factory
function createPool(
    address tokenA,
    address tokenB,
    uint24 fee
) external override returns (address pool) {
    ...
    require(getPool[token0][token1][fee] == address(0));
    pool = IPancakeV3PoolDeployer(poolDeployer).deploy(address(this), token0, token1,
fee, tickSpacing);
    getPool[token0][token1][fee] = pool;
    // populate mapping in the reverse direction, deliberate choice to avoid the cost of
comparing addresses
    getPool[token1][token0][fee] = pool;
    emit PoolCreated(token0, token1, fee, tickSpacing, pool);
}
```

However, attackers can calculate the token address in advance and then front-running to create a pool, causing addToken to be revert.

Recommendation

If the pool has already been created, there is no need to call createV3Pool().

Status

This issue has been resolved by the team.

2.3 Informational Findings

11. Lack of _disableInitializers

Severity: Informational	Category: Business Logic
Target: - contracts/Factory.sol	

Description

The `Factory` logic contract lacks `_disableInitializers` to disable the initialization of the contract, allowing anyone to call the initialize function in `Factory` logic contract which increases the attack surface.

Recommendation

We recommend adding `_disableInitializers` in the constructor of the `Factory` contract.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
Activity.sol	ffcaf0a86cb2fb4887685efaf9d4b67ef8547567
Factory.sol	1366b380b2f03a55f2d1a8b50701e79398423f83
LibActivityTime.sol	1a77f8770fdce4ebe1b5d2ad2cdc745e29b65995
FourMemeERC20.sol	2be7c5a5b409bacda81c8a88334eb60437b3c3ad

And we audited the files that introduced new features:

File	SHA-1 hash
Activity.sol	f798cdc171217f836eb3bd8ad755b69a646669f9
Factory.sol	38ca57bafab087f4716b94c3881433bccc7effc0

And we audited the files that include fixes:

File	SHA-1 hash
Activity.sol	cad99151dc54222f2de7288c780e1d32f3836809
SwapHelper.sol	649a2964333cc5aa2381b580803bb7666c5906ad