SALUS SECURITY

APR 2025

CODE SECURITY ASSESSMENT

UPTOP

Overview

Project Summary

- Name: Uptop - CakeTower
- Platform: The BSC Blockchain
- Language: Solidity
- Address: 0x5519259a4ce265e62448c839d829a37a2d281e3e
- Fixed Address:
  - Proxy: 0xE59b2F3848D8b29FD939657d56E44673113b6C1f
  - Implementation: 0x5913142e771dd50a39e02de66c45a25637d1f5e4
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Uptop - CakeTower
Version	v2
Type	Solidity
Dates	Apr 23 2025
Logs	Apr 22 2025; Apr 23 2025

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	4
Total Low-Severity issues	1
Total informational issues	2
Total	9

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	Repeat exit to get loss	High	Business Logic	Resolved
2	Misconfigurated ownership during deployment prevents access to owner-restricted functions	High	Configuration	Resolved
3	Missing slippage check	Medium	Front-running	Resolved
4	Assuming the pool is vulnerable to price manipulation attacks	Medium	Front-running	Resolved
5	TokenBonus update is incorrect	Medium	Business Logic	Resolved
6	Inconsistent token address validation in createCakeTower leads to transaction reversion	Medium	Business Logic	Resolved
7	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky External Calls	Resolved
8	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
9	Incorrect deadline in liquidity functions	Informational	Business logic	Resolved

2.2 Notable Findings

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

1. Repeat exit to get loss	
Severity: High	Category: Business Logic
Target: - contracts/CakeTower.sol	

Description

This is a user exit mechanism. When a user exits, all of their floors will be marked as inactive, liquidity will be removed and ETH will be returned. If there is a loss, tokens will be compensated in proportion. But, this compensation may be repeated.

contracts/CakeTower.sol:L336-L362

```
function exit() public {
    uint256[] memory floors = userTowers[msg.sender];
    for (uint256 i = 0; i < floors.length; i++) {
        uint256 floor = floors[i];
        towers[floor].isActive = false;
    }
    ...
    if (amountETH >= fee * floors.length) {
       ...
    } else {
        uint256 loss = fee * floors.length - amountETH;
        uint256 lossToken = amountToken.mulDiv(loss, amountETH, Math.Rounding.Floor);
        IERC20(token).safeTransfer(msg.sender, lossToken);
        tokenBalance += amountToken - lossToken;
    }
}
```

Attach Scenario:

1. Attacker builds two floors and exits when `amountETH < fee * floors.length` `lossToken` contains two floors.
2. Attacker builds a new floor and exits. Due to the total number of floors owned by the attacker is 3, `lossToken` contains 3 floors. (But the previous two floor's losses have been sent.)
3. Repeat step 2 until the Tower is hollowed out.

Recommendation

Compensate only active floors.

```
function exit() public {
    uint256[] memory floors = userTowers[msg.sender];
+   uint activeFloors = 0;
    for (uint256 i = 0; i < floors.length; i++) {
        uint256 floor = floors[i];
```

```
-           towers[floor].isActive = false;
+           if(towers[floor].isActive) {
+              towers[floor].isActive = false;
+              activeFloors++;
+           }
}
...
-      if (amountETH >= fee * floors.length) {
+      if (amountETH >= fee * activeFloors) {
    tokenBalance += amountToken;
} else {
-           uint256 loss = fee * floors.length - amountETH;
+           uint256 loss = fee * activeFloors - amountETH;
```

Status

The team has resolved this issue in the logic contract address 0xE59b...6C1f.

2. Misconfigurated ownership during deployment prevents access to owner-restricted functions	
Severity: High	Category: Configuration
Target: - contracts/CakeTower.sol - contracts/CakeTowerFactory.sol	

Description

When the CakeTower contract is deployed by the Factory contract, the CakeTower constructor sets the owner to `msg.sender`. At deployment time, `msg.sender` corresponds to the Factory contract rather than the intended end user.

contracts/CakeTower.sol:L88-L111

```
constructor() {
    …
    owner = msg.sender;
    …}
```

Consequently, the owner-restricted functions—such as `settle()`, `addTokenBonus()`, and `addTokenBalance()`—can only be called by the Factory contract. However, the Factory contract does not provide any mechanism to invoke these functions on the deployed CakeTower instance, effectively locking these owner-only functions from being used by the proper administrator or the end user.

contracts/CakeTower.sol:L403-L423

```
function addTokenBonus(uint256 amount) public {
    require(msg.sender == owner, "Not the contract owner");
    …
}

function addTokenBalance(uint256 amount) public {
    require(msg.sender == owner, "Not the contract owner");
    …
}

function settle() public onlyOwner {
    IERC20(token).safeTransfer(msg.sender, tokenBonus + tokenBalance);
}
```

Recommendation

Set the correct user as the owner.

Status

The team has resolved this issue in the logic contract address 0xE59b...6C1f.

3. Missing slippage check	
Severity: Medium	Category: Front-running
Target: - contracts/CakeTower.sol	

Description

The contract's liquidity management functions—including `buildFloor()`, `addLiquidity()` and `removeLiquidity()`—do not implement adequate slippage protection. Specifically, when calling external functions on PancakeSwap's router (such as `addLiquidityETH` and `removeLiquidityETH`), the minimum acceptable token and ETH amounts (i.e., `amountTokenMin` and `amountETHMin`) are set to zero.

contracts/CakeTower.sol:L255-L282

```
function addLiquidity(uint256 buildFee, uint256 ethR, uint256 tokenR)
   internal
   returns (uint256 amountToken, uint256 amountETH, uint256 liquidity)
{
   ...
   (amountToken, amountETH, liquidity) =
IPancakeRouter01(pancakeRouter).addLiquidityETH{value: buildFee}(
       token, amountTokenMin, 0, 0, address(this), block.timestamp + 20 minutes
   );
   tokenBalance -= amountToken;
   return (amountToken, amountETH, liquidity);
}

function removeLiquidity(uint256 liquidity) internal returns (uint256 amountToken,
uint256 amountETH) {
   ...
   (amountToken, amountETH) = IPancakeRouter01(pancakeRouter).removeLiquidityETH(
       token, liquidity, 0, 0, address(this), block.timestamp + 20 minutes
   );
   return (amountToken, amountETH);
}
```

An attacker can manipulate the pool reserves through flashloan to capture a more favorable position. Also users might end up receiving lower-than-expected rewards or even lose funds.

Recommendation

Implement a slippage check.

Status

The team has resolved this issue in the logic contract address [0xE59b...6C1f](0xE59b...6C1f).

4. Assuming the pool is vulnerable to price manipulation attacks	
Severity: Medium	Category: Front-running
Target: - contracts/CakeTowerFactory.sol - contracts/CakeTower.sol	

Description

The `createCakeTower` function only checks if the token is a valid ERC20 contract. The existence of the corresponding trading pair on PancakeSwap and the status of the liquidity pool were not verified.

contracts/CakeTowerFactory.sol

```
function createCakeTower(...) public payable {
    require(token == address(0) || IERC20(token).totalSupply() > 0, "Invalid token
address");
    // ...
}
```

Attach Scenario:

1. Alice created a new Tower of TokenA without a BNB/TokenA pool.
2. The attacker created a pool of BNB/TokenA with a very high proportion such as one BNB:one million TokenA with a very small amount of token.
3. When the attacker calls `buildFloor`, the `getTokenRate` will return the ratio of the pool manipulated by the attacker. When adding liquidity, the tower needs to pay a large amount of tokens or even be emptied.
4. The attacker returns to the pool and pays a small amount of BNB to empty most of tokenA.

Recommendation

Make sure the pool exists before createTower.

Status

The team has resolved this issue in the logic contract address 0xE59b...6C1f.

5. TokenBonus update is incorrect

Severity: Medium	Category: Business logic
Target: - contracts/CakeTower.sol	

Description

According to the `blindBoxReward` and `distributeFloorBonus` function code, after five new floors are created, the creator can receive 60% of the rewards of floorBounus, and the remaining 40% will be distributed as blind box rewards.

contracts/CakeTower.sol:L264-L323

```
function blindBoxReward(uint256 floor, uint256 randomness) internal {
    …
    uint256 bonus = towers[floor].bonus.mulDiv(DEFAULT_BLIND_REWARD_RATE, 100,
Math.Rounding.Floor);
    …
}
function distributeFloorBonus(uint256 floor) internal {
    uint256 times = DEFAULT_FLOOR_REWARD_DIV;
    while (floor > 0 && times > 0) {
        Floor memory f = towers[floor];
        awardFloor(floor, f.bonus.mulDiv(DEFAULT_FLOOR_REWARD_RATE, 100 *
DEFAULT_FLOOR_REWARD_DIV, Math.Rounding.Ceil), 1, floor);
    …    }
}
```

However, `CakeTower` is completely deducted during the execution of `_buildFloor`. Obviously, There will be floors that are not lucky, and the last five floors, these bonuses will not be distributed.

contracts/CakeTower.sol:L196-L209

```
function _buildFloor(uint256 index, string memory message) internal {
    uint256 floorBounus = calculateFloorBonus(index);
    …
    tokenBonus -= floorBounus;
    …
}
```

Recommendation

Only deduction when rewards are distributed.

Status

The team has resolved this issue in the logic contract address [0xE59b...6C1f](0xE59b...6C1f).

6. Inconsistent token address validation in createCakeTower leads to transaction reversion	
Severity: Medium	Category: Business logic
Target: - contracts/CakeTowerFactory.sol	

Description

In the Factory contract's `createCakeTower()` function, a user is allowed to specify a token address, including the possibility of providing the zero address. However, in the same function, if the token address is `address(0)`, transferFrom will revert.

contracts/CakeTowerFactory.sol:L74-L112

```
function createCakeTower(
   …
) public payable {
   require(msg.value >= createFee, "Insufficient fee");
   require(token == address(0) || IERC20(token).totalSupply() > 0, "Invalid token
address");
   …
   require(IERC20(token).transferFrom(msg.sender, foo, tokenBonus + tokenBalance),
"Transfer failed");
   emit CakeTowerCreated(
       foo, msg.sender, icon, title, message, token, createFee, bonusBase, tokenBonus,
tokenBalance, bnbPrice
   );
}
```

Recommendation

If a zero address is intended to represent a valid special token, implement conditional logic to bypass the `transferFrom()` call when token is `address(0)` and handle that case appropriately.

Status

The team has resolved this issue in the logic contract address [0xE59b...6C1f](0xE59b...6C1f).

7. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	
Severity: Low	Category: Risky External Calls
Target: - contracts/CakeTowerFactory.sol - contracts/CakeTower.sol	

Description

contracts/CakeTowerFactory.sol:L80

```
require(IERC20(token).transferFrom(msg.sender, foo, tokenBonus + tokenBalance),
"Transfer failed");
```

contracts/CakeTower.sol:L368, L376

```
IERC20(token).transferFrom(msg.sender, address(this), amount);
IERC20(token).transferFrom(msg.sender, address(this), amount);
```

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

The team has resolved this issue in the logic contract address 0xE59b...6C1f.

2.3 Informational Findings

8. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/CakeTowerFactory.sol	

Description

The CakeTowerFactory contract uses OwnableUpgradeable which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

contracts/CakeTowerFactory.sol:L9-L113

```
contract CakeTowerFactory is OwnableUpgradeable {
   ...
}
```

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

9. Incorrect deadline in liquidity functions	
Severity: Informational	Category: Business logic
Target: - contracts/CakeTower.sol	

Description

In the liquidity management functions (`addLiquidity()` and `removeLiquidity()`), the deadline parameter for PancakeRouter01 function calls is set as `block.timestamp + 20 minutes`. This fixed extension of 20 minutes is redundant since it does not provide any meaningful guarantee for transaction finality or execution timing.

contracts/CakeTower.sol:L255-L282

```
function addLiquidity(uint256 buildFee, uint256 ethR, uint256 tokenR)
   internal
   returns (uint256 amountToken, uint256 amountETH, uint256 liquidity)
{
   …
   (amountToken, amountETH, liquidity) =
IPancakeRouter01(pancakeRouter).addLiquidityETH{value: buildFee}(
       token, amountTokenMin, 0, 0, address(this), block.timestamp + 20 minutes
   );
   tokenBalance -= amountToken;
   return (amountToken, amountETH, liquidity);
}

function removeLiquidity(uint256 liquidity) internal returns (uint256 amountToken,
uint256 amountETH) {
   …
   (amountToken, amountETH) = IPancakeRouter01(pancakeRouter).removeLiquidityETH(
       token, liquidity, 0, 0, address(this), block.timestamp + 20 minutes
   );
   return (amountToken, amountETH);
}
```

Recommendation

Remove the code `+ 20 minutes`.

Status

The team has resolved this issue in the logic contract address [0xE59b...6C1f](0xE59b...6C1f).

Appendix

Appendix 1 - Files in Scope

This audit covered the following files from address 0x5519259a4Ce265E62448C839D829A37A2d281e3E:

File	SHA-1 hash
contracts/CakeTowerFactory.sol	eda0a79267411d45a2ebd7bd5e1c46c56ed74ab0
contracts/CakeTower.sol	82d67302cb9ceab29707cbb236cd34d6fc4c7ff8
contracts/interfaces/IPancakeFactory.sol	969677dfcbc8e488d4322850aca5568719a7ba35
contracts/interfaces/ICakeTower.sol	cc7c9b1c3908c7063381f9c0636d5c2c3f333faf