SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

WTECHMY

Overview

Project Summary

- Name: Wtechmy - Sp-smart-contract
- Platform: The BSC Blockchain
- Language: Solidity
- Repository:
  - https://bitbucket.org/wtechmy/sp_smart_contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Wtechmy - Sp-smart-contract
Version	v4
Type	Solidity
Dates	May 28 2024
Logs	May 15 2024; May 19 2024; May 22 2024; May 28 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	5
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

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 caliber, 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	Centralization risk	Low	Centralization	Acknowledged
2	Presale start time needs to be greater than block.timestamp	Low	Configuration	Resolved
3	Owner withdrawal functions lack presale end time validation	Low	Business Logic	Resolved
4	Use BNB-USD oracle to mitigate precision issues	Low	Configuration	Resolved
5	Missing checks on chainlink feed return value	Low	Data Validation	Resolved
6	Gas optimization suggestions	Informational	Gas Optimization	Resolved
7	Events are not indexed	Informational	Logging	Resolved

2.2 Notable Findings

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

1. Centralization risk	
Severity: Low	Category: Centralization
Target: - smartContract/preSales.sol - smartContract/withdraw.sol	

Description

The `` `preSales` `` contract has privileged accounts. These privileged accounts can:
1. Withdraw all Unsold tokens before the presale end-time using the `` `removeStuckToken()` ``.
2. Changing the token exchange rate so that buyers/holders suffer losses.
3. Modify the token sale time and end it at any time.

The `` `withdraw` `` contract has privileged accounts. These privileged accounts change the limit of different categories to destroy the proportion of token distribution using the `` `updateLimit()` `` function.

If privileged accounts' private key or admin's is compromised, an attacker can steal all the tokens in the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

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 and states that the owner function is intended for emergency use only.

2. Presale start time needs to be greater than block.timestamp	
Severity: Low	Category: Configuration
Target: - smartContract/preSales.sol	

Description

The `` `setPresaleTime()` `` function only enforces that `` `_start < _end` `` but does not require that the chosen start time is strictly in the future (i.e. `` `>= block.timestamp` ``). This allows the contract owner to inadvertently set the presale's start timestamp to a past value.

smartContract/preSales.sol:L77-L82

```
function setPresaleTime(uint256 _start, uint256 _end) external onlyOwner {
   require(_start < _end, "Invalid presale period");
   presaleStart = _start;
   presaleEnd = _end;
   emit setPresaleTimeSPEvent(_start, _end, "setPresaleTimeSPEvent");
}
```

Recommendation

Add a validation check to ensure the presale start time is not set in the past.

Status

This issue has been resolved by the team with commit 32de529.

3. Owner withdrawal functions lack presale end time validation	
Severity: Low	Category: Business Logic
Target: - smartContract/preSales.sol	

Description

The `withdrawBNB()`, `withdrawUSDT()`, and `withdrawUnsoldTokens()` functions can be called by the contract owner at any time, including during the active presale window. None of these functions checks that `block.timestamp` is after `presaleEnd`, so the owner could prematurely withdraw the collected funds or remaining sale tokens while users are still purchasing.

smartContract/preSales.sol:L133-L150

```
function withdrawBNB() external onlyOwner {
    uint256 balance = address(this).balance;
    require(balance > 0, "No BNB to withdraw");
    (bool success,) = payable(owner()).call{value: balance}("");
    require(success, "BNB withdrawal failed");
}

function withdrawUSDT() external onlyOwner {
    uint256 balance = usdtToken.balanceOf(address(this));
    require(balance > 0, "No USDT to withdraw");
    usdtToken.safeTransfer(owner(), usdtToken.balanceOf(address(this)));
}

function withdrawUnsoldTokens() external onlyOwner {
    uint256 unsoldTokens = slothspepeToken.balanceOf(address(this));
    require(unsoldTokens > 0, "No tokens to withdraw");
    slothspepeToken.safeTransfer(owner(), unsoldTokens);
}
```

Recommendation

Add a presale-end time check to each withdrawal function to ensure they can only be executed after the sale window closes.

Status

This issue has been resolved by the team with commit [32de529](32de529).

4. Use BNB-USD oracle to mitigate precision issues	
Severity: Low	Category: Configuration
Target: - smartContract/preSales.sol	

Description

In the `preSales` contract, by using the usd-bnb oracle to calculate `tokensToBuy`. However, Solidity can only use integers to simulate decimals (no native decimals). There will be a loss of precision for the division operation. Obviously, there are two division operations in the process of calculating `tokensToBuy`. After testing, there will be a deviation of about 1%. Secondly, through chainlink feeds, we can see that BNB-USD Oracle is updated more frequently than USD-BNB Oracle, so it is more recommended to use.

smartContract/preSales.soll:L103-L140

```
function bnbRate() public view returns (uint256) {
    uint256 USDToBNBPrice = 1000000000000000000 / getBNBUSD();
    uint256 SlothspepeToBNB = tokenUSDPrice / USDToBNBPrice;
    return SlothspepeToBNB;
}

function getBNBUSD() public view returns (uint256) {
    (, int256 price, , , ) = bnbUsd.latestRoundData();
    require(price >= 0, "Chainlink price is negative"); // 防止负数转成超大 uint
    return uint256(price);
}

function usdtRate() public view returns (uint256) {
    return tokenUSDPrice;
}

function buyWithBNB() external payable onlyDuringPresale {
    ...
    uint256 tokensToBuy = (msg.value * ONE) / bnbRate();
    ...
}
```

Recommendation

Using BNB-USD Oracles to mitigate accuracy issues and be more reliable.

Status

This issue has been resolved by the team with commit c753b64.

5. Missing checks on chainlink feed return value	
Severity: Low	Category: Data Validation
Target: - smartContract/preSales.sol	

Description

In the `` `getBNBUSD()` `` function, the implementation retrieves the price using the `` `latestRoundData()` `` function. However, it does not perform validation to check whether the returned price data is stale.

smartContract/preSales.sol:L109-L113

```
function getBNBUSD() public view returns (uint256) {
   (, int256 price,,,) = bnbUsd.latestRoundData();
   require(price >= 0, "Chainlink price is negative");
   return uint256(price);
}
```

Recommendation

Add checks to ensure the returned price data is fresh, for example:

```
function getBNBUSD() public view returns (uint256) {
-   (, int256 price,,,) = bnbUsd.latestRoundData();
+   (uint80 quoteRoundID, int256 price,, uint256 quoteTimestamp, uint80
quoteAnsweredInRound) = bnbUsd.latestRoundData();
+   require(quoteAnsweredInRound >= quoteRoundID, "Stale price!");
+   require(quoteTimestamp != 0, "Round not complete!");
+   require(block.timestamp - quoteTimestamp <= VALID_TIME_PERIOD);
    require(price >= 0, "Chainlink price is negative");
    return uint256(price);
}
```

Status

This issue has been resolved by the team with commit 046e6cf.

2.3 Informational Findings

6. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - smartContract/preSales.sol	

Description

Both `buyWithBNB()` and `buyWithUSDT()` include a check:

smartContract/preSales.sol:L96-L144

```
function buyWithBNB() external payable onlyDuringPresale {
   require(msg.value > 0, "BNB amount required");
   require(depositStatus == true, "Invalid deposit status");
   require(block.timestamp >= presaleStart && block.timestamp <= presaleEnd, "Invalid
Presales Time");
   ...
}

function buyWithUSDT(uint256 _amount) external onlyDuringPresale {
   require(_amount > 0, "USDT amount required");
   require(depositStatus == true, "Invalid deposit status");
   require(block.timestamp >= presaleStart && block.timestamp <= presaleEnd, "Invalid
Presales Time");
   ...
}
```

Even though they are already guarded by the `onlyDuringPresale` modifier, which performs the identical check.

smartContract/preSales.sol:L49-L52

```
modifier onlyDuringPresale() {
   require(block.timestamp >= presaleStart && block.timestamp <= presaleEnd, "Presale
not active");
   _;
}
```

This duplication increases bytecode size and unnecessarily consumes extra gas on each purchase transaction.

Recommendation

Remove the extra checks from both functions.

Status

This issue has been resolved by the team with commit 32de529.

7. Events are not indexed	
Severity: Informational	Category: Logging
Target: - smartContract/preSales.sol - smartContract/withdraw.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. The emitted events are not indexed, making off-chain scripts such as front-ends of dApps difficult to filter the events efficiently.

In the `withdraw` contract, the `WithdrawSPEvent` event is not indexed.

In the `preSales` contract, the `TokensPurchasedSPEvent` event is not indexed.

Recommendation

Consider adding the indexed keyword in the `WithdrawSPEvent` and `TokensPurchasedSPEvent` events.

Status

This issue has been resolved by the team with commit 32de529.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
smartContract/preSales.sol	83a09ec6a9de66c29385ab1c1b2e0b34b486c5eb
smartContract/withdraw.sol	4be4fdd3eb1a2d5399f4ace252bcfc6264a89098

And we audited the commit 32de529 that introduced new features wtechmy/sp_smart_contract repository:

File	SHA-1 hash
smartContract/preSales.sol	1fda448bebdc73b3d5ea4f764985cbe2eabe6d53
smartContract/withdraw.sol	877b25fb4a9096df69ad8995e36c2acdad62dc46