AstraSec

SonicSphere Sale Security Audit Report

May 31, 2025

Contents

1 Introduction

1.1 About SonicSphere Sale Contract

The Sale contract manages the initial distribution of oSPHERE tokens. It offers a fair and equal opportunity for all participants to purchase oSPHERE at a fixed price of 0.5 S per oSPHERE. The sale has two phases: first, only whitelisted addresses can participate, then it opens to everyone. There are no individual purchase limits - all participants can buy as many tokens as they want at the same price. The sale requires a minimum of 5,000,000 S to proceed, with a maximum cap of 10,000,000 S. If the minimum amount isn't reached, all participants receive a full refund of their S. If the sale succeeds, the SonicSphere team can withdraw the funded S for audits and project development. After the sale concludes, participants can claim their oSPHERE tokens from this contract.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/Heesho/sonic-sphere/blob/main/contracts/Sale.sol
- Commit: 3b60acf893bdac0d626192f0c88591b1055c3e8

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/Heesho/sonic-sphere/blob/main/contracts/Sale.sol
- Commit: 1ffdc88a1ee3d321cda28d5773d6e8e2cd27568a

1.3 Revision History

Version	Date	Description
v1.0	May 31, 2025	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Sale contract. Throughout this audit, we identified a total of 3 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	–	–	–	–
Low	2	1	–	1
Informational	1	–	–	1
Total	3	1	–	2

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [L-1] Improved Transfer of Native Token using Address.sendValue()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Sale.sol	Gas Limitation	Medium	Low	Addressed

The refundFor() function in the Sale contract uses Solidity's payable.transfer() method to send native token refunds to users when the Sale contract gets into the Refund state. The payable.transfer() method imposes a fixed gas stipend of 2300, which is insufficient in certain scenarios. With EIP-1884 (introduced in the Istanbul hard fork), the gas costs of certain opcodes, such as SLOAD, have increased. This can cause the recipient's fallback or receive function to consume more than 2300 gas, resulting in a failed native token transfer. Consequently, users may be unable to receive their refunds, rendering the contract's refund mechanism unreliable.

openzeppelin-contracts - Sale.sol

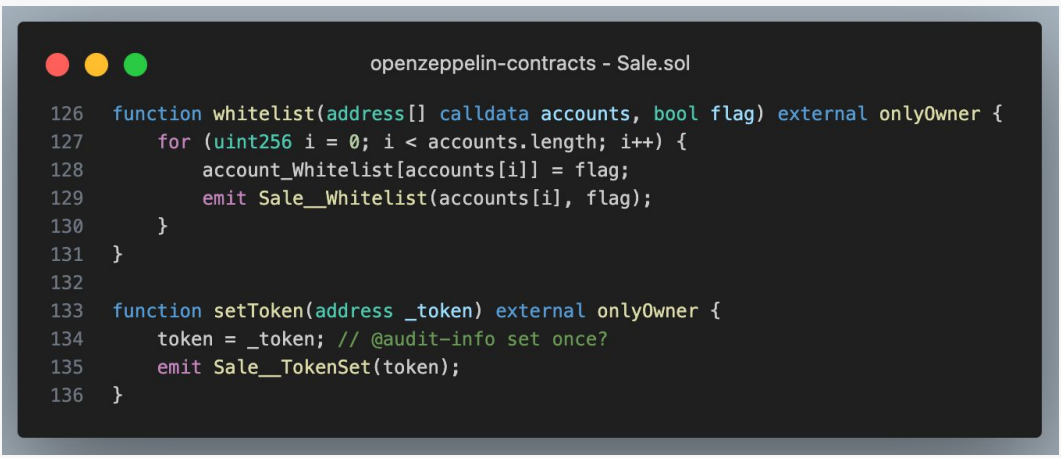
```
108  function refundFor(address account) external nonReentrant {
109      if (account == address(0)) revert Sale__InvalidAccount();
110      if (state != State.Refund) revert Sale__NotRefund();
111      if (account_Amount[account] == 0) revert Sale__InvalidRefund();
112
113      uint256 refund = account_Amount[account] - account_Refund[account];
114      if (refund <= 0) revert Sale__InvalidRefund();
115
116      account_Refund[account] += refund;
117      totalRefund += refund;
118
119      payable(account).transfer(refund);
120
121      emit Sale__Refund(account, refund);
122  }
```

Remediation Update the refundFor() function to replace the payable.transfer() method with OpenZeppelin's Address.sendValue() function, which forwards all available gas to the recipient, removing the 2300 gas limitation. This ensures that the native token transfer can succeed even if the recipient's fallback or receive function requires more gas.

3.3.2 [L-2] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Sale.sol	Security	Medium	Low	Acknowledged

The Sale contract grants significant protocol-wide control to a single privileged owner account, enforced by the onlyOwner modifier. The owner has the authority to execute critical actions, including setting the sale token address, managing whitelist accounts, updating the sale state, and withdrawing funds from the contract. This centralization of power introduces substantial risks to the protocol's decentralization ethos, as the owner can unilaterally alter the protocol's behavior or misappropriate funds. If the owner's private key is compromised, or if the owner acts maliciously, it could lead to protocol manipulation, fund drainage, or disruption of the sale process, undermining user trust and the integrity of the system.



Remediation To mitigate centralization risks, consider implementing a multi-signature wallet (e.g., using Gnosis Safe) or a decentralized governance mechanism to manage critical actions. Additionally, introduce a time-lock mechanism for sensitive actions to provide users with advance notice and the opportunity to react to changes. If full decentralization is not feasible, ensure the owner's role is transparently documented, and consider transferring ownership to a secure, community-controlled entity over time to align with decentralization principles.

Response By Team This issue has been acknowledged by the team.

3.3.3 [I-1] Inefficient Gas Usage in purchaseFor()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Sale.sol	Gas Optimization	Informational	N/A	Addressed

The purchaseFor() function allows users to purchase tokens with native token, ensuring the total amount does not exceed MAX_CAP. On line 83, the function calculates the new total amount as total = totalAmount + amount to check against MAX_CAP. However, on line 87, it redundantly recalculates the same value totalAmount + amount when updating totalAmount. This redundant computation wastes gas, as the previously computed total value could be reused. While this does not introduce any security risk, it does reduce the contract's gas efficiency and increases transaction costs for users.

Remediation Optimize gas usage by eliminating the redundant computation. Instead of re-calculating totalAmount + amount, directly update totalAmount using the already computed total value. This reduces the number of operations, lowering the gas cost of the purchaseFor() function.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI