expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

SonicxFunPool

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	2
Medium	0
Low	3
Informational	0

Centralization Risks

Owner Privileges	Description
Can Owner Set Taxes >25% ?	Detected
Owner needs to enable trading ?	Not Detected
Can Owner Disable Trades ?	Not Detected
Can Owner Mint ?	Not Detected
Can Owner Blacklist ?	Not Detected
Can Owner set Max Wallet amount ?	Not Detected
Can Owner Set Max TX amount ?	Not Detected

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**High Risk Detected**
Audit Date	**13 April 2025**

CONTRACT DETAILS

Contract Address:
0xBbe710B32411b83A6D5A24F296EC3813864822C8

Contract Name: SonicxPool

Blockchain: Sonic

Contract Type: ERC-20

Contract Creator:
0x965fB7b0D6ffe64729eEba6943B86ef3edB1262c

Compiler Version: v0.8.20+commit.a1b79de6

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- Manual Review: The code has undergone a line-by-line review by the Ace team.
- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.
- Slither: The code has undergone static analysis using Slither.

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

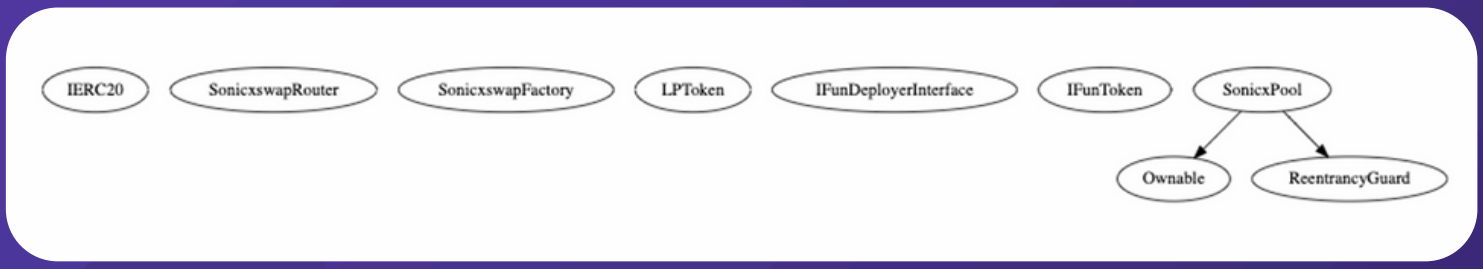
Low Risk

Issues on this level are minor details and warning that can remain unfixed.

Informational

Issues on this level are minor details and warning that can remain unfixed.

INHERITANCE TREE



POINTS TO NOTE

- The owner has administrative privileges over the launchpad
- The owner can transfer ownership

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

HIGH RISK FINDING

Centralization – Erroneous ETH Reserve Reset

Severity: High

Overview:
This operation in the "buyTokens" (L411) function sets the ETH reserve to zero after adding liquidity, which is a programming error. Instead of reducing the reserve by a specific amount, it subtracts the entire reserve from itself, resulting in zero. This breaks the entire bonding curve mechanism since future price calculations depend on non-zero reserves. Users who interact with the contract after this operation would experience incorrect pricing or complete transaction failures.

```
token.pool.reserveETH -= token.pool.reserveETH;
```

Recommendation: Remove this line entirely or replace it with code that properly accounts for the remaining ETH after liquidity addition, such as setting it to zero explicitly if that's the intention.

HIGH RISK FINDING

Centralization – Missing Fee Parameters
Severity: High

Description:

These functions allow the owner to set arbitrary fee percentages without any upper limits. The feePer is used to calculate fees during token trades while lpfee determines LP token distribution. Without caps, the owner could potentially set fees to extreme values (up to 65,535 for uint16), effectively preventing users from trading or extracting nearly all value from transactions.

```
function updateteamFeeper(uint16 _newFeePer) public onlyOwner { feePer =
_newFeePer; }
function updateLpfee(uint16 _lpfee) public onlyOwner { lpfee = _lpfee; }
```

Recommendation: Implement strict upper bounds for all fee parameters to prevent malicious or accidental excessive fee setting. Use require statements to enforce these limits. consider implementing a timelock for fee changes to give users adequate notice before fees are modified.

LOW RISK FINDING

Centralization – Missing Zero-Address Validation

Severity: Low

Suggestion:
The addDeployer function doesn't verify that the provided address is non-zero. Setting the zero address as a deployer would be meaningless but would still consume gas and could lead to confusion. This pattern is repeated in other address-setting functions, representing a broader issue where critical address parameters lack proper validation throughout the contract.

function CreateFun(FunParameters memory params) public payable { }

Recommendation: Add zero-address validation in all functions that set address parameters.

LOW RISK FINDING

Centralization – Missing Event Emissions
Severity: Low

Description:

None of the administrative functions emits events when changing critical parameters, making it impossible to track changes off-chain.

function addDeployer(address _deployer) public onlyOwner { allowedDeployers[_deployer] = true; } function updateteamFeeper(uint16 _newFeePer) public onlyOwner { feePer = _newFeePer; } function updateLpfee(uint16 _lpfee) public onlyOwner { lpfee = _lpfee; } function addDeployer(address _deployer) public onlyOwner { allowedDeployers[_deployer] = true; } function removeDeployer(address _deployer) public onlyOwner { allowedDeployers[_deployer] = false; } function updateImplementation(address _implementation) public onlyOwner { require(_implementation != address(0)); implementation = _implementation; } function updateteamFeeper(uint16 _newFeePer) public onlyOwner { } function updateLpfee(uint16 _lpfee) public onlyOwner { } }

Recommendation: Add events for all state-changing functions.

LOW RISK FINDING

Centralization – Incorrect Import Path Syntax

Severity: Low

Overview:
The contract uses backslashes instead of forward slashes in import paths, which will cause compilation errors on most platforms.

import "@openzeppelin\contracts\proxy\Clones.sol"; import "@openzeppelin\contracts\utils\ReentrancyGuard.sol"; import "@openzeppelin\contracts\access\Ownable.sol"; import "@chainlink\contracts\src\v0.8\shared\interfaces\AggregatorV3Interface.sol";

Recommendation: Replace backslashes with forward slashes

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.