expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

SonicxFunDeployer

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	1
Medium	2
Low	4
Informational	3

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:
0xE381b1Fc43C291c29f77eFAaa144aCC2DE301d5C

Contract Name: SonicxDeployer

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

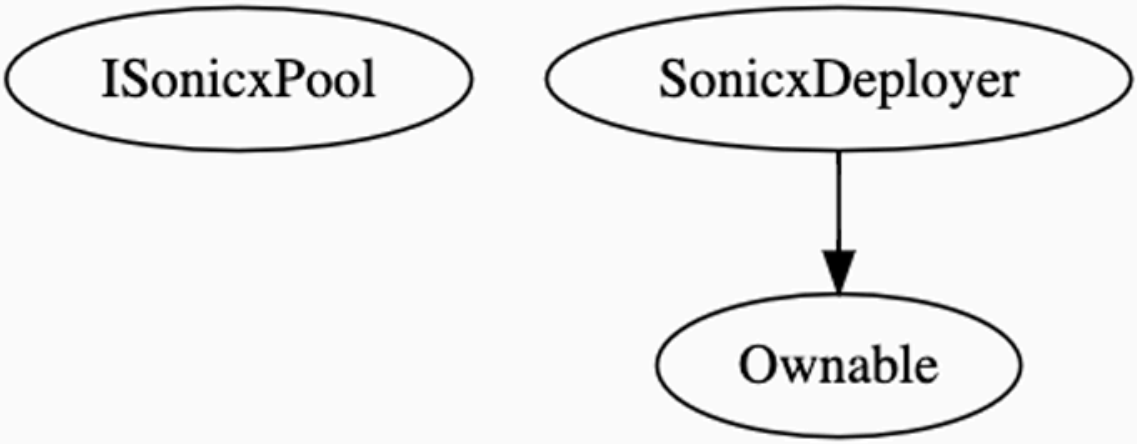
Issues on this level are minor details and warning that can remain unfixed.

Informational

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

expelee

INHERITANCE TREE



POINTS TO NOTE

- The owner can update supply values and reserve ETH values
- The owner can lock/unlock the token supply
- The owner can add/disable/enable routers and base tokens
- The owner can update the fun pool address
- The owner can update the list thresholds
- The owner can enable/disable LP burning
- The owner can withdraw funds in an emergency
- 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 – Uncapped Fee Parameters

Severity: HIGH

Description:
The contract allows the owner to set any arbitrary fee amounts with no upper limits. This could be misused to extract excessive fees from users or effectively block token creation.

function updateTeamFee(uint256 _newTeamFeeInWei) public onlyOwner { teamFee = _newTeamFeeInWei; } function updateownerFee(uint256 _newOwnerFeeBaseTenK) public onlyOwner { ownerFeePer = _newOwnerFeeBaseTenK; }

Recommendation: Implement reasonable upper bounds for all fee parameters. The fees cannot exceed more than 25%.

MEDIUM RISK FINDING

Centralization – Unsafe ETH Transfer Pattern

Severity: Medium

Description:

The emergencyWithdraw function uses the outdated transfer() method which limits gas to 2300 units. This can cause the function to fail if the recipient is a contract with complex receive logic.

function emergencyWithdraw() public onlyOwner { uint256 balance = address(this).balance; payable(owner()).transfer(balance); }

Recommendation: Use the safer call pattern with successful verification.

MEDIUM RISK FINDING

Centralization – Lack of Input Validation

Severity: Medium

Description:
No validation is performed on user-supplied token parameters like name, symbol, and totalSupply, potentially allowing the creation of tokens with invalid or malicious configurations.

function CreateFun(FunParameters memory params) public payable { }

Recommendation: Add basic parameter validation.

LOW RISK FINDING

Centralization – Missing Zero Address

Severity: Low

Suggestion:

Critical functions such as the constructor, updateFunPool, addRouter, and addBaseToken do not validate against zero address inputs. Setting core contract addresses to zero would render the contract permanently inoperable.

constructor(address _sonicxPool) Ownable(msg.sender) { sonicxPool = _sonicxPool; } function updateFunPool(address _newfunPool) public onlyOwner { sonicxPool = _newfunPool; } function addRouter(address _routerAddress) public onlyOwner { } function addBaseToken(address _baseTokenAddress) public onlyOwner {}

Suggestion: Add explicit zero address checks for all 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 updateTeamFee(uint256 _newTeamFeeInWei) public onlyOwner {
teamFee = _newTeamFeeInWei;    } function updateownerFee(uint256
_newOwnerFeeBaseTenK) public onlyOwner {        ownerFeePer =
_newOwnerFeeBaseTenK;    } function addBaseToken(address
_baseTokenAddress) public onlyOwner {
require(!baseAdded[_baseTokenAddress], "already added");
baseAdded[_baseTokenAddress] = true;        baseValid[_baseTokenAddress] =
true;        baseStorage[baseCount] = _baseTokenAddress;        baseCount++;    }
function updateFunPool(address _newfunPool) public onlyOwner {
sonicxPool = _newfunPool;      } function updateListThreshold(uint256
_newListThreshold) public onlyOwner {        listThreshold = _newListThreshold;    }
```

Recommendation: Add events for all state-changing functions.

LOW RISK FINDING

Centralization – No Slippage Protection

Severity: Low

Description:
The anti-snipe mechanism sets minTokens to 0 when buying tokens, providing no slippage protection. This exposes users to MEV attacks like sandwiching.

```
if (params.antiSnipe) {     ISonicxPool(sonicxPool).buyTokens{value:
params.amountAntiSnipe}(funToken, 0;
```

Recommendation: Implement proper slippage protection by calculating minimum expected tokens.

LOW RISK FINDING

Centralization – Incorrect Import Path Syntax

Severity: Low

Description:

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

import "@openzeppelin\contracts\utils\ReentrancyGuard.sol"; import "@openzeppelin\contracts\token\ERC20\IERC20.sol"; import "@openzeppelin\contracts\access\Ownable.sol";

Recommendation: Replace backslashes with forward slashes.

INFORMATIONAL FINDING

Centralization – Inconsistent Function Naming
Severity: Information

Description:

Function names don't follow Solidity conventions and are inconsistent within the contract. Some functions start with uppercase letters, others use inconsistent capitalization.

function CreateFun(FunParameters memory params) public payable function updateownerFee(uint256 _newOwnerFeeBaseTenK) public onlyOwner function enableBasetoken(address _baseTokenAddress) public onlyOwner

Recommendation: Standardize function naming using camelCase.

INFORMATIONAL FINDING

Centralization – Unused code
Severity: Information

Description:

Several state variables are declared but never used in the contract logic, wasting gas and potentially causing confusion.

uint256 public supplyValue = 1000000000 ether; bool public supplyLock = true;

Recommendation: Remove unused code.

INFORMATIONAL FINDING

Centralization – Floating Pragma
Severity: Information

Description:

The contract uses a floating pragma (^0.8.20), which could lead to inconsistent behaviour if compiled with different compiler versions.

pragma solidity ^0.8.20;

Recommendation: Lock the pragma to a specific version.

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.