SALUS SECURITY

MAR 2023

CODE SECURITY ASSESSMENT

LSDX

Overview

Project Summary

- Name: LSDx
- Version: commit b6f968c
- Platform: Ethereum
- Language: Solidity
- Contract Address Set:
  LSDCoin StakingPoolFactory EthStakingPool
  stETHStakingPool frxETHStakingPool rETHStakingPool LPStakingPool
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	LSDx
Version	v1
Type	Solidity
Dates	Mar 19 2023
Logs	Mar 19 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	8
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	Centralization risk	Low	Centralization	Unresolved
2	Floating compiler version	Informational	Configuration	Unresolved
3	Missing zero address checks	Informational	Data Validation	Unresolved
4	Inconsistent import is used	Informational	Code Quality	Unresolved
5	Inconsistent type of unsigned integers	Informational	Code Quality	Unresolved
6	Unnecessary payable modifier	Informational	Code Quality	Unresolved
7	Redundant code	Informational	Redundancy	Unresolved
8	Gas optimization suggestions	Informational	Gas Optimization	Unresolved
9	State variable that could be declared immutable	Informational	Gas Optimization	Unresolved

2.2 Notable Findings

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

1. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/LsdCoin.sol - contracts/EthStakingPool.sol	

Description

The upgradeable proxy pattern is used in the LsdCoin contract. The proxy admin controls the upgrade mechanism to upgradeable proxies, they can change the respective implementations. And should the admin’s private key be compromised, an attacker could upgrade the logic contract to execute their own malicious logic on the proxy state.

The MINTER_ROLE is authorized to mint LSD Coins. In the event that the private key for this role is breached, the attacker can utilize the mint function to generate a substantial amount of LSD Coins.

The EthStakingPool has a function called withdrawELRewards() which enables privileged roles to take out Ether from the contract. Should the admin's private key be compromised, and the user has not withdrawn their stake Ether by the end of a stake round, an intruder can use the withdrawELRewards() function to take the user's Ether from the contract.

Recommendation

Consider transferring the privileged roles to multi-sig accounts.

2.3 Informational Findings

2. Floating compiler version

Severity: Informational	Category: Configuration
Target: - all	

Description

```
pragma solidity ^0.8.9;
```

The LSDx contracts use a floating compiler version ^0.8.9.

Using a floating pragma ^0.8.9 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

3. Missing zero address checks

Severity: Informational	Category: Data Validation

Target:
- contracts/StakingPoolFactory.sol
- contracts/StakingPool.sol

Description

contracts/StakingPoolFactory.sol:L39-L45

```
constructor(
  address _rewardsToken,
  address _nativeTokenWrapper
) Ownable() {
  rewardsToken = _rewardsToken;
  nativeTokenWrapper = _nativeTokenWrapper;
}
```

contracts/StakingPool.sol:L38-L48

```
constructor(
  address _rewardsDistribution,
  address _rewardsToken,
  address _stakingToken,
  uint256 _durationInDays
) {
  rewardsToken = IERC20(_rewardsToken);
  stakingToken = IERC20(_stakingToken);
  rewardsDistribution = _rewardsDistribution;
  rewardsDuration = _durationInDays.mul(3600 * 24);
}
```

It is considered a security best practice to verify addresses against the zero-address during constructor or setting. However, this precautionary step is absent for address variables in the constructors.

Recommendation

Consider adding zero-address checks for address variables in the constructors.

4. Inconsistent import is used

Severity: Informational	Category: Code Quality
Target: - contracts/StakingPoolFactory.sol	

Description

contracts/StakingPoolFactory.sol:L4

```
import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
```

contracts/StakingPool.sol:L9

```
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

The highlighted **IERC20.sol** is employed in StakingPoolFactory to execute the rewardsToken call. However, in StakingPool, the same rewardsToken call is carried out by using the second highlighted section of the **SafeERC20.sol** import, which creates a discrepancy between the two.

SafeERC20 is considered superior to regular ERC20 because it guarantees the safety of ERC20 function calls by verifying the boolean return values of ERC20 operations, and reversing the transaction if they are unsuccessful. It also checks the use of non-standard ERC20 tokens that lack boolean return values. Furthermore, SafeERC20 includes functions to increase or decrease allowances, which mitigates the possibility of front-running attacks that could arise from using the approve() function in standard ERC20. As a result, it is recommended to use the SafeERC20 library with ERC20 token.

Recommendation

Consider using SafeERC20 in both the StakingPoolFactory and StakingPool contracts to improve code consistency.

5. Inconsistent type of unsigned integers

Severity: Informational	Category: Code Quality
Target: - contracts/StakingPool.sol	

Description

contracts/StakingPool.sol:L140

```
uint balance = rewardsToken.balanceOf(address(this));
```

The highlighted line uses **uint** for 256 bits unsigned integers, while **uint256** is used in other places for the same type.

Using an explicit **uint256** type instead of the **uint** type can improve code readability.

Recommendation

Consider using uint256 instead of uint to improve code consistency and readability

6. Unnecessary payable modifier

Severity: Informational	Category: Code Quality
Target: - contracts/StakingPoolFactory.sol	

Description

contracts/StakingPoolFactory.sol:L87

```
StakingPool(payable(address(info.poolAddress))).withdrawELRewards(to);
```

contracts/StakingPoolFactory.sol:L102

```
StakingPool(payable(address(info.poolAddress))).notifyRewardAmount(rewardsAmount);
```

Since neither the withdrawELRewards function nor the notifyRewardAmount function in StakingPool involves receiving Ether, the addition of the payable modifier is unnecessary.

Recommendation

Consider removing the payable modifier.

7. Redundant code

Severity: Informational	Category: Redundancy
Target: - contracts/StakingPool.sol	

Description

contracts/StakingPool.sol:L8

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
```

SafeMath is used to check underflow and overflow for arithmetic operations. However, since Solidity version 0.8.0, arithmetic operations revert on underflow and overflow by default. Since the bounce project uses a Solidity version no less than 0.8.0, it is unnecessary to use the **SafeMath** library.

contracts/StakingPool.sol:L12

```
import "./lib/CurrencyTransferLib.sol";
```

The import of library CurrencyTransferLib.sol is redundant, it is not used in the StakingPool contract.

Recommendation

Consider removing the redundant code.

8. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization

Target:
- contracts/RewardsDistributionRecipient.sol
- contracts/StakingPoolFactory.sol
- contracts/StakingPool.sol

Description

contracts/StakingPool.sol:L24-L25

```
uint256 public periodFinish = 0;
uint256 public rewardRate = 0;
```

Since the values default to zero, the initialization can be removed to save gas.

contracts/StakingPoolFactory.sol:L47

```
function getStakingPoolAddress(address stakingToken) public virtual view returns
(address) {
```

contracts/StakingPoolFactory.sol:L53

```
function getStakingTokens() public virtual view returns (address[] memory) {
```

contracts/StakingPoolFactory.sol:L60

```
function deployPool(address stakingToken, uint256 startTime, uint256
roundDurationInDays) public onlyOwner {
```

contracts/StakingPoolFactory.sol:L90

```
function addRewards(address stakingToken, uint256 rewardsAmount) public onlyOwner {
```

These functions are not called inside the contract, thus, the visibility can be changed to external to save gas.

contracts/RewardsDistributionRecipient.sol

```
require(msg.sender == rewardsDistribution, "Caller is not RewardsDistribution
contract");
```

Strings in Solidity are handled in 32-byte chunks. A string longer than 32 bytes will consume more gas. Shortening this string saves gas.

contracts/StakingPool.sol:L87

```
require(amount > 0, "Cannot stake 0");
```

The contract uses the 'require' statement for a conditional check, but starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Using custom errors can reduce both gas usage and bytecode size.

Recommendation

Consider removing unnecessary variable initialization.

Consider changing the visibility of getStakingPoolAddress, getStakingTokens, deployPool, addRewards function to external.

Consider reducing the length of the revert messages to 32 bytes or less.

Consider using custom errors to replace the revert messages.

9. State variable that could be declared immutable

Severity: Informational	Category: Gas Optimization

Target:
- contracts/StakingPoolFactory.sol
- contracts/StakingPool.sol
- contracts/EthStakingPool.sol

Description

contracts/StakingPoolFactory.sol:L15-L16

```
address public rewardsToken;
address public nativeTokenWrapper;
```

contracts/StakingPool.sol:L22-L23

```
IERC20 public rewardsToken;
IERC20 public stakingToken;
```

contracts/EthStakingPool.sol:L16

```
IWETH public weth;
```

To reduce gas costs, the state variables rewardsToken, nativeTokenWrapper, stakingToken, and weth could be declared as immutable since their values are fixed after the contracts have been deployed.

Recommendation

Consider adding the immutable modifier to rewardsToken, nativeTokenWrapper, and weth state variables.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in commit b6f968c:

File	SHA-1 hash
contracts/EthStakingPool.sol	8df030e76886992e5159437f3033cbf800bfd29c
contracts/LsdCoin.sol	eaa60365d1e0b588785be513d096909d4f4dd61f
contracts/RewardsDistributionRecipient.sol	e8d56c900c313a9d5f7c12c5e057480cd9330b62
contracts/StakingPool.sol	4e34d084ee2966bed6908e25c7f90283cb6cbbe7
contracts/StakingPoolFactory.sol	6b6000124c9965692edc9a4b8090d99e60884a80
contracts/interfaces/IStakingPool.sol	0d7736656bacd5c6650805b409d381add6971d6a
contracts/interfaces/IWETH.sol	a023498b5f142b977d5cad063994a396c71662c9
contracts/lib/CurrencyTransferLib.sol	94b44bdf78df91339ce2c995f909429cae928532