SALUS SECURITY

JUL 2024

CODE SECURITY ASSESSMENT

ORA

Overview

Project Summary

- Name: ORA - staking-contract
- Platform: Ethereum
- Language: Solidity
- Repository:
  - https://github.com/ora-io/staking-contract-audit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ORA
Version	v5
Type	Solidity
Dates	Jul 13 2024
Logs	Jun 21 2024, Jul 02 2024; Jul 03 2024; Jul 05 2024; Jul 13 2024

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	3
Total Low-Severity issues	6
Total informational issues	2
Total	13

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	Lack of access control for stakeWithPermit()	High	Access Control	Resolved
2	Revisit stETH pool's implementation	High	Business Logic	Resolved
3	Possible DoS attack via permit()	Medium	Denial of Service	Resolved
4	Use low-level call instead of transfer()	Medium	Denial of Service	Resolved
5	Centralization risk	Medium	Centralization	Mitigated
6	Snapshots should be relatively consistent	Low	Business Logic	Acknowledged
7	Incorrect initialization for ORAStakePoolBase_ERC7641	Low	Configuration	Resolved
8	Lack of timely update of the vault's current TVL	Low	Business Logic	Resolved
9	The grace period might be bypassed	Low	Business Logic	Resolved
10	Missing check pool is valid	Low	Data Validation	Resolved
11	Inheritance order warning and no storage gap	Low	Configuration	Resolved
12	Typo error	Informational	Code Quality	Resolved
13	Gas optimization suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

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

1. Lack of access control for stakeWithPermit()	
Severity: High	Category: Access Control
Target: - src/ORAStakePool_StETH.sol - src/ORAStakePool_OLM.sol	

Description

In the `` `ORAStakePool_StETH.sol` `` and `` `ORAStakePool_OLM.sol` ``, the `` `stakeWithPermit()` `` function is used to support the ERC20 permit.
However, the `` `stakeWithPermit()` `` function lacks proper access control, allowing users to bypass the router and call the `` `stakeWithPermit()` `` function directly.

The impact of this vulnerability includes:
1. The check in `` `ORAStakeRouter.sol` `` that ensures the vault's current TVL does not exceed its maximum TVL can be bypassed.
2. Even when the router contract is suspended, users can still deposit tokens into the contract.

Recommendation

Consider adding the `` `onlyRouter` `` modifier to the `` `stakeWithPermit()` `` function.

Status

This issue has been resolved by the team with commit f4ddf14.

2. Revisit stETH pool’s implementation	
Severity: High	Category: Business Logic
Target: - src/ORAStakePool_StETH.sol	

Description

1. In the use of StETH's `transferFrom()` function, the third parameter should represent the amount rather than shares.

2. In the `ORAStakePool_StETH` contract, the shares minted for stakers equal the staked StETH amount, causing users to lose their deserved StETH rewards..

Recommendation

Consider recording the shares transferred by the staker to the contract and calculating the corresponding amount of StETH returned based on the shares transferred.

Status

This issue has been resolved by the team with commit f4ddf14.

3. Possible DoS attack via permit()	
Severity: Medium	Category: Denial of Service
Target: - src/ORAStakePool_StETH.sol - src/ORAStakePool_OLM.sol	

Description

The `` `ORAStakePool_StETH` `` and `` `ORAStakePool_OLM` `` contracts support ERC20 permits through the stakeWithPermit() function.

However, malicious users can front-run by calling permit() and using up the signature. This prevents normal users from successfully calling stakeWithPermit() This prevents normal users from successfully calling stakeWithPermit() because the signature is already used up, causing the permit() function to fail.

Attach Scenario

1. Alice signs one signature to approve the contract and calls stakeWithPermit() to stake some stETH tokens.
2. Bob monitors this and calls the permit() function for Alice using Alice's signature via frontrun.
3. Alices' stakeWithPermit() transaction will be reverted because the signature has been used and is not valid anymore.

Recommendation

Suggest checking the allowance before calling the permit() function.

Status

This issue has been resolved by the team with commit f4ddf14.

4. Use low-level call instead of transfer()	
Severity: Medium	Category: Denial of Service
Target: - src/ORAStakePoolBase.sol	

Description

When users want to claim withdrawal, the Ora protocol will use transfer() function to return the ether.

The transfer() operation sends only 2,300 units of gas with this operation. As a result, if the recipient is an AA wallet, or multisig smart contract, the transfer() might be reverted because of out of gas. Stakers' funds will be stuck in the contract.

Recommendation

Consider using a low-level call to replace the transfer() function.

Status

This issue has been resolved by the team with commit f4ddf14.

5. Centralization risk	
Severity: Medium	Category: Centralization
Target: - src/ORAStakeRouter.sol - src/ORAStakePoolBase.sol	

Description

In ORAStakeRouter and other pool contracts, there exists a privileged owner role. This owner has authority over key operations such as updating critical parameters and upgrading implementation contracts to modify service logic.

If the owner's private key were compromised, an attacker could exploit this access to withdraw all tokens by upgrading the implementation contracts.

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

The team has moved owner permissions to the multi-sig account 0xF8aF2Bd9B83a4C9D1c79b2B637426e8e29EAeda4.

6. Snapshots should be relatively consistent	
Severity: Medium	Category: Business Logic
Target: - src/utils/ERC7641Upgradeable.sol - src/ORAStakePoolBase_ERC7641.sol	

Description

Typically, users stake OLM tokens in the ORAStakePool_OLM contract. Users can claim their corresponding rewards after the ORAStakePool_OLM contract has claimed rewards from the OLM contract.
The vulnerability arises if the ORAStakePool_OLM contract's snapshot does not accurately reflect the state of the OLM token. This discrepancy can result in staking users losing their rewards.

Recommendation

Provide a mechanism like a monitor to ensure that the snapshot times of the two contracts are consistent.

Status

This issue has been acknowledged by the team. And the team states that they will make use of scripts to make sure olm token's snapshot and stake pool's reward claim in one transaction.

7. Incorrect initialization for ORAStakePoolBase_ERC7641	
Severity: Low	Category: Configuration
Target: - src/ORAStakePoolBase_ERC7641.sol	

Description

When initializing the `ORAStakePoolBase_ERC7641` contract, the `ERC20Upgradeable` contract is left uninitialized.

This will cause the ERC20's name and symbol to be empty.

Recommendation

Initialize ERC20Upgradeable parameter via __ERC20__init(), similar to the ORAStakePoolBase contract.

Status

This issue has been resolved by the team with commit f4ddf14.

8. Lack of timely update of the vault’s current TVL	
Severity: Low	Category: Business Logic
Target: - src/ORAStakeRouter.sol	

Description

In the router contract, each vault's current TVL is recorded. The vulnerability arises when the current TVL of vaults is not promptly updated upon calling updatePool() or removePool(). Although there is a syncTVL() function intended to synchronize the current TVL, there are potential time gaps where vaults' TVL is not synchronized.

When a vault's TVL is not promptly updated, we may encounter the following issues:

1. The check for the vault's maxTVL may be bypassed.

2. The claimWithdraw() function could revert because the recorded current TVL is less than the actual current TVL.

Recommendation

Timely update vault’s current TVL when updating or removing the pool.

Status

This issue has been resolved by the team with commit f4ddf14.

9. The grace period might be bypassed	
Severity: Low	Category: Business Logic
Target: - src/ORAStakeRouter.sol	

Description

In the router contract, there is one withdrawGracePeriod mechanism. When stakers want to withdraw funds, stakers need to request withdrawal, and after the grace period, stakers can withdraw their funds.

The vulnerability is when users request to withdraw, the pool contract checks whether the request amount exceeds the balanceOf(user) or not. However, stakers can send multiple withdrawal requests, which will lead to the total request withdrawal amount being larger than the balanceOf(user). The Withdrawal grace period might be bypassed because of this vulnerability.

Recommendation

Total request withdrawal amount should be less than the balance of the staker.

Status

This issue has been resolved by the team with commit f4ddf14.

10. Missing check pool is valid	
Severity: Low	Category: Data Validation
Target: - src/ORAStakeRouter.sol	

Description

In the `ORAStakeRouter` contract, it has a validPoolOnly check in `claimWithdraw()` function for one pool. However, it is missing the validPoolOnly check in claimWithdraw() function for the pool array.

Recommendation

Consider adding the validPoolOnly check.

Status

This issue has been resolved by the team with commit f4ddf14.

11. Inheritance order warning and no storage gap	
Severity: Low	Category: Configuration
Target: - src/ORAStakePoolBase_ERC7641.sol - src/utils/ERC7641Upgradeable.sol	

Description

The ERC7641Upgradeable contract is an upgradeable contract and it implements a custom initialization mechanism. Hence, it is important to avoid storage collisions in the contract of upgrading the implementation. Due to this, adding the storage gap to the ERC7641Upgradeable contract is necessary.

Recommendation

Consider reversing the inheritance order and adding storage gaps to the ERC7641Upgradeable contract.

Status

This issue has been resolved by the team with commit f4ddf14.

2.3 Informational Findings

12. Typo error	
Severity: Informational	Category: Code Quality
Target: - src/ORAStakeRouter.sol	

Description

The `parseRequest()` function is conventionally used to analyze and extract data from incoming requests. However, in this case, the function is used to pause the Withdraw Request. This creates an inconsistency between the function name and its actual purpose.

Recommendation

Consider renaming the function.

Status

This issue has been resolved by the team with commit f4ddf14.

13. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - src/ORAStakePoolBase.sol - src/ORAStakeRouter.sol	

Description

1. Memory reading saves more gas than storage reading multiple times when the state is not changed. So caching the storage variables in memory and using the memory instead of storage reading is effective. So cache array length outside of the loop can save gas.
src/ORAStakeRouter.sol:L100

```
for (uint256 i = 0; i < pools.length; i++) {
```

src/ORAStakeRouter.sol:L113

```
for (uint256 i = 0; i < vaults[vaultId].pools.length; i++) {
```

src/ORAStakeRouter.sol:L169

```
for (uint256 i = 0; i < pools.length; i++) {
```

src/ORAStakeRouter.sol:L191

```
for (uint256 i = 0; i < vaultPools.length; i++) {
```

src/ORAStakePoolBase.sol:L108

```
for (uint256 i = 0; i < vaults[vaultId].pools.length; i++) {
```

2. There are redundant checks in the `` `ORAStakeRouter` `` contract `` `SyncTVL()` `` function. So removing the redundant checks can save gas.

Recommendation

Consider using the above suggestions to save gas.

Status

This issue has been resolved by the team with commit f4ddf14.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit b20e975:

File	SHA-1 hash
src/ORAStakeRouter.sol	9e3f322adef2287d63c02af432e96bcff99966b3
src/ORAStakePoolBase.sol	25c3cc7bb2f5c6c13ca29619d310fc8bbea7d3fe
src/ORAStakePoolBase_ERC7641.sol	d09a12c80fcbcbe137a60ac899fb23c4407da8b4
src/ORAStakePool_StETH.sol	45747d1b539b64f5f124277a1cc604d3eb976229
src/ORAStakePool_OLM.sol	6f15efe2bb15c0b6bd7192ef3da299e797c00984
src/ORAStakePool_StakeStoneETH.sol	f0b2ddddf0cc6913404f894a3d1687563b84e1a4
src/ORAStakePool_ETH.sol	91572d184c281400e4e8d9b21af8f0ab302e5496
src/utils/ERC7641Upgradeable.sol	3d4d26e56988675a18f034164fec88da20a889a0
src/utils/ERC20SnapshotUpgradeable.sol	34d721c59c15be0844d2b8932670aac708ef7702