SALUS SECURITY

JUL 2024

CODE SECURITY ASSESSMENT

PARALLEL FINANCE

Overview

Project Summary

- Name: Parallel Finance - Pac contract
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/parallel-finance/
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Parallel Finance - Pac contract
Version	v2
Type	Solidity
Dates	Jul 09 2024
Logs	Jul 05 2024, Jul 09 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	4
Total	10

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 slippage control on zapIntoLp()	High	Business Logic	Resolved
2	Pause functionality unavailable	Medium	Business Logic	Resolved
3	Centralization Risk	Medium	Centralization	Acknowledged
4	Rewards may be locked in the contract	Medium	Business Logic	Resolved
5	Lack of array length validation	Low	Data Validation	Resolved
6	Implementation contract could be initialized by everyone	Low	Code Quality	Resolved
7	Wrong data in the event RewardDeposited	Informational	Logging	Resolved
8	Use of floating pragma	Informational	Compiler	Resolved
9	Uninformative error message	Informational	Logging	Resolved
10	Ineffective use of the deadline parameter	Informational	Data Validation	Resolved

2.2 Notable Findings

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

1. Lack of slippage control on zapIntoLp()	
Severity: High	Category: Business Logic
Target: - contracts/plp-staking/PLPLocking.sol	

Description

In the `` `zapIntoLp()` `` function, users can add their Ether/WETH and Pac tokens to the router pool to receive LP tokens and lock these LP tokens in a single transaction. However, there is a vulnerability: the slippage control parameters, `` `amountAMin` `` and `` `amountBMin` ``, are set to 0. This makes the transaction highly susceptible to Miner Extractable Value (MEV) manipulation, potentially forcing users to add liquidity at an unfair price.

plp-staking/PLPLocking.sol:L660-L671

```
function zapIntoLp(
   uint256 pacAmountDesired,
   uint256[] calldata lockIndexes,
   uint256 lockConfigIndex
) external payable whenNotPaused returns (uint256 liquidity) {
    ...
   (, , liquidity) = thrusterRouter.addLiquidity(
       pacToken,
       weth,
       pacAmountDesired,
       ethAmountRequired,
       0,
       0,
       address(this),
       block.timestamp + 60 * 60
   );
   _lockLP(liquidity, user, lockConfigIndex, true);
    ...
}
```

Recommendation

Provide slippage control parameters as the input parameters.

Status

This issue has been resolved by the team with commit ad9fb8e.

2. Pause functionality unavailable

Severity: Medium	Category: Business Logic
Target: - contracts/plp-staking/PLPLocking.sol	

Description

The `PLPLocking` contract inherits from the `PausableUpgradeable` contract. Key functions include the `whenNotPaused` modifier, ensuring that these functions can only be executed when the contract is not in pause mode.

contracts/BridgeV2.sol:L16-20

```
contract PLPLocking is
   Initializable,
   PausableUpgradeable,
   OwnableUpgradeable,
   IPLPLocking
{
    ...
}
```

However, there are no public functions to pause or unpause the contract. As a result, the contract owner cannot enable or disable the pause mode.

Recommendation

It is necessary to add external functions to the contract to control the pause mode.

Status

This issue has been resolved by the team with commit ba98c10.

3. Centralization Risk

Severity: Medium	Category: Centralization
Target: - contracts/plp-staking/PLPLocking.sol	

Description

In the `PLPLocking.sol`, there is a privileged Owner role. This role has the ability to:

- Pause the contract
- Set locking config
- Add/remove reward tokens

If it is an EOA account, any compromise of its private key could drastically affect the project. For example, attackers could pause the contract or set the locking token to block the current contract's logic if they gain access to the private key.

Recommendation

We recommend transferring the admin role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

4. Rewards may be locked in the contract	
Severity: Medium	Category: Business Logic
Target: - contracts/plp-staking/PLPLocking.sol	

Description

In depositRewards(), the contract owner deposits rewards for LP stakers. These rewards are distributed from lastUpdateTime to periodEndTime.

If no LP tokens are locked during a specific time slot, the associated rewards cannot be claimed by anyone. Additionally, the owner has no way to retrieve these unclaimed rewards, causing them to be permanently locked in the contract

plp-staking/PLPLocking.sol:L318-L350

```
function depositRewards(
   address[] calldata tokens,
   uint256[] calldata amount
) external onlyOwner {
       ...
       uint256 reward = amount[i];
       if (reward > 0) {
           IERC20(token).safeTransferFrom(
               msg.sender,
               address(this),
               reward
           );
           tokensRewardInfo[token].balance += reward;
           tokensRewardInfo[token].periodEndTime = block.timestamp;
           tokensRewardInfo[token].rewardPerSecond =
               (reward * REWARD_PRECISION) /
               (block.timestamp - tokensRewardInfo[token].lastUpdateTime);

       }
       ...
   }
}
```

Recommendation

Add one sweep function to sweep unused rewards.

Status

This issue has been resolved by the team with commit ba98c10.

5. Lack of array length validation	
Severity: Low	Category: Data validation
Target: - contracts/plp-staking/PLPLocking.sol	

Description

The depositRewards function accepts two arrays as parameters but does not verify that they are of the same length. Implementing this check will prevent potential issues where the array containing amounts is longer than the array containing tokens.

plp-staking/PLPLocking.sol:L660-L671

```
function depositRewards(
   address[] calldata tokens,
   uint256[] calldata amount
) external onlyOwner {
   uint256 tokensLength = tokens.length;
   for (uint256 i; i < tokensLength; ) {
       address token = tokens[i];
       if (tokensRewardInfo[token].lastUpdateTime == 0) revert NotExist();

       uint256 reward = amount[i];
       if (reward > 0) {
           ...
       }
       unchecked {
           i++;
       }
   }
}
```

Recommendation

Consider adding checks at the beginning of the function to compare the lengths of the `tokens` and `amounts` arrays. This will help ensure they are of equal length and prevent potential issues.

Status

This issue has been resolved by the team with commit ba98c10.

6. Implementation contract could be initialized by everyone

Severity: Low	Category: Code Quality
Target: - contracts/plp-staking/PLPLocking.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the `` `initialize()` `` function in `` `PLPLocking` ``'s implementation contract.

plp-staking/PLPLocking.sol:L67-L75

```
constructor() {}

function initialize(
   address _pLPLockersTable,
   address _timeVesting,
   address _pacToken,
   address _thrusterRouter,
   address _weth
) public initializer {
```

Recommendation

To prevent the implementation contract from being used, consider invoking the `` `_disableInitializers()` `` function in the constructor of the `` `PLPLocking` `` contract to automatically lock it when it is deployed.

Status

This issue has been resolved by the team with commit ba98c10.

2.3 Informational Findings

7. Wrong data in event RewardDeposited	
Severity: Informational	Category: Logging
Target: - contracts/plp-staking/PLPLocking.sol	

Description

The contract includes a `RewardDeposited` event, which is emitted by the `depositRewards` function.

plp-staking/PLPLocking.sol:L335-L339

```
function depositRewards(
    address[] calldata tokens,
    uint256[] calldata amount
) external onlyOwner {
        ...
        emit RewardDeposited(
            address(this),
            token,
            reward,
            block.timestamp
         );
```

interfaces/IPLPLocking.sol:L60-L65

```
event RewardDeposited(
    address indexed user,
    address rewardToken,
    uint256 amount,
    uint256 periodEndTime
);
```

The first field of the `RewardDeposited` event is `user`, but the function passes the address of the contract itself instead of the address of the user (owner) who called the function.

Recommendation

Consider emitting the correct event.

Status

This issue has been resolved by the team with commit ba98c10.

8. Use of floating pragma	
Severity: Informational	Category: Compiler
Target: - All files	

Description

Protocol uses a floating compiler version ^0.8.0.

Using a floating pragma ^0.8.0 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. 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.

Status

This issue has been resolved by the team with commit ba98c10.

9. Uninformative error message	
Severity: Informational	Category: Logging
Target: - contracts/plp-staking/PLPLocking.sol	

Description

The `addRewardTokens` function accepts an array of token addresses as its input parameter. It checks that each address is neither a zero address nor has been added previously.

plp-staking/PLPLocking.sol:L116-L127

```
function addRewardTokens(
   address[] calldata _rewardTokens
) external onlyOwner {
   uint256 length = _rewardTokens.length;
   if (length == 0) revert InvalidRewardTokens();

   for (uint256 i; i < length; ) {
       address _rewardToken = _rewardTokens[i];
       if (_rewardToken == address(0)) revert AddressZero();

       if (tokensRewardInfo[_rewardToken].lastUpdateTime != 0)
           revert TokenAlreadySet();
```

The `TokenAlreadySet()` error does not provide specific information about which token has already been added previously.

Recommendation

Add a parameter to the error that includes the index of the element with the address that caused the error.

Status

This issue has been resolved by the team with commit ba98c10.

10. Ineffective use of the deadline parameter	
Severity: Informational	Category: Data Validation
Target: - contracts/plp-staking/PLPLocking.sol	

Description

In the `zapIntoLp` function, liquidity is added using the `addLiquidity` function, which includes a `deadline` parameter to ensure the transaction is completed before a specified time. However, the current implementation always passes an incorrect value to the `addLiquidity` function: it uses the current timestamp plus 60*60 seconds. This means the check will always pass, making the deadline parameter ineffective.

plp-staking/PLPLocking.sol:L619-L660

```
 function zapIntoLp(
       uint256 pacAmountDesired,
       uint256[] calldata lockIndexes,
       uint256 lockConfigIndex
   ) external payable whenNotPaused returns (uint256 liquidity) {
   ...
   (, , liquidity) = thrusterRouter.addLiquidity(
           pacToken,
           weth,
           pacAmountDesired,
           ethAmountRequired,
           0,
           0,
           address(this),
           block.timestamp + 60 * 60
    );
}
```

Recommendation

Consider enhancing the `zapIntoLp()` function by adding an additional parameter named `deadline`, which allows the caller to manually specify the deadline timestamp for the liquidity addition.

Status

This issue has been resolved by the team with commit ba98c10.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit f06e5bf:

File	SHA-1 hash
TimeVesting.sol	1857cf2a41b9b2aeb556442f2082c98f2ed60696
PLPLocking.sol	0c0a17749385234fc2a61208747ed50dd0e5e5f5