Competitive Security Assessment

Lagrange_Update_2

May 28th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3′s Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.
The comprehensive examination and auditing scope includes:
• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
• Verify the code base is compliant with the most up-to-date industry standards and security best practices.
• Comprehensive line-by-line manual code review of the entire codebase by industry experts.
The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

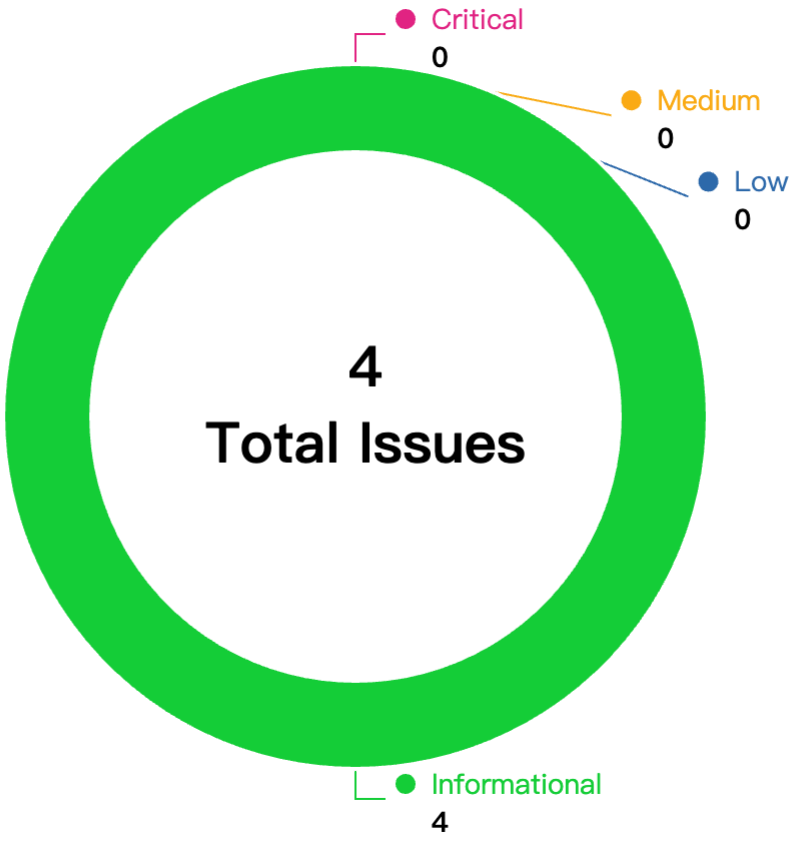
Overview

Project Name	Lagrange_Update_2
Language	Solidity
Codebase	• https://github.com/Lagrange-Labs/lagrange-contracts.git • audit version - fbd5f0f2867c23d39d10072acbb90a67e4c20486 • final version - aa157d054df1100b1ce56eee2e1ea58db5c0d3c7
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/protocol/LagrangeCommittee.sol	f2530180e081b125e1010921bb56f39a3476a3e5d3a953043b004ee257d4f5a3
contracts/protocol/LagrangeService.sol	2f40aeded61bd5557603bbaf5e11f2221b2f3ca3aef4041f27fe5b91dd35450e
contracts/interfaces/ILagrangeCommittee.sol	3c061201871b3ad4141444ce469872795cfc23f80e08dc15e0098e8fe0f77965
contracts/interfaces/ILagrangeService.sol	0833e0e47d329284a0c7649ba4310a4e8ef3b92d74b9162e643a35f1b85e521f

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LU2-1	The function `updateSignAddress` allows setting signer address, even if the operator is not registered.	Logical	Informational	Fixed	1nc0gn170
LU2-2	Missing event trigger	Code Style	Informational	Fixed	Bryce
LU2-3	Consider processing the array before it is written to the storage to save gas	Gas Optimization	Informational	Fixed	ethprinter
LU2-4	Code optimization	Code Style	Informational	Fixed	Bryce

LU2-1:The function `updateSignAddress` allows setting signer address, even if the operator is not registered.

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	1nc0gn170

Code Reference

- code/contracts/protocol/LagrangeCommittee.sol#L134

```
134: function updateSignAddress(address operator, address newSignAddress) external onlyService {
```

Description

1nc0gn170:
The function **`updateSignAddress`** is meant to update the **`signAddress`** of the corresponding operator. However, the function lacks a check to ensure whether the operator is registered or not.

```
    function updateSignAddress(address operator, address newSignAddress) external onlyService {
        operatorsStatus[operator].signAddress = newSignAddress;
    }
```

It is not ideal to allow setting a value before it even exists, so it is better to restrict this scenario.

Recommendation

1nc0gn170:

```
    // Updates an operator's sign address
    function updateSignAddress(address operator, address newSignAddress) external onlyService {
+       require(operatorsStatus[operator].blsPubKeys.length != 0, "Operator is not registered.");
        operatorsStatus[operator].signAddress = newSignAddress;
    }
```

Client Response

client response for 1nc0gn170: Fixed. commit-aa157d054df1100b1ce56eee2e1ea58db5c0d3c7.
`LagrangeCommittee.updateSignAddress` is called only from **`LagrangeService.updateSignAddress`** which includes validating operator whitelisted.

LU2-2:Missing event trigger

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Bryce

Code Reference

- code/contracts/protocol/LagrangeCommittee.sol#L134-L136

```
134: function updateSignAddress(address operator, address newSignAddress) external onlyService {
135:         operatorsStatus[operator].signAddress = newSignAddress;
136:     }
```

Description

Bryce: In the `updateSignAddress` function, modifications are made to the critical data `signAddress` in the contract, but does not trigger the corresponding event, which is not conducive to obtaining on-chain data.

Recommendation

Bryce: It is recommended to declare the corresponding event and trigger it in the function.

Client Response

client response for Bryce: Fixed. commit-aa157d054df1100b1ce56eee2e1ea58db5c0d3c7

LU2-3:Consider processing the array before it is written to the storage to save gas

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	ethprinter

Code Reference

- code/contracts/protocol/LagrangeCommittee.sol#L128-L130

```
128: for (uint256 i = count; i < _length; i++) {
129:             operatorsStatus[operator].blsPubKeys.pop();
130:         }
```

Description

ethprinter: In the **`LagrangeCommittee`**::**`removeBlsPubKeys()`** method, a memory variable **`blsPubKeys`** is first created and goes through a series of processes before being written to storage. However, after lines 128-130, there is still a **`pop`** operation performed on **`blsPubKeys`**. This is less efficient than directly performing operations on the previous memory variable. A test script as follows shows that operating directly on memory before writing can save a significant amount of gas.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Counter {
    mapping(address => uint[]) public operatorsStatus;

    uint[] public largeArray;

    constructor() {
        for (uint i = 0; i < 1000; i++) {
            largeArray.push(1);
        }
    }


    function test_removeBlsPubKeys_1() external {
        operatorsStatus[address(0x1234)] = largeArray;
        uint256 _length = largeArray.length;
        for (uint256 i; i < _length; i++) {
            operatorsStatus[address(0x1234)].pop();
        }
    }

    function test_removeBlsPubKeys_2() external {
        uint[] memory array = largeArray;
        uint256 _length = array.length;
        for (uint256 i; i < _length; i++) {
            array[i] = 0;
        }

        operatorsStatus[address(0x1234)] = array;
    }
}
```

result show as flows:

```
Ran 2 tests for test/Counter.t.sol:Counter
[PASS] test_removeBlsPubKeys_1() (gas: 19810804)
[PASS] test_removeBlsPubKeys_2() (gas: 4559989)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 9.27ms (5.97ms CPU time)
```

Recommendation

ethprinter: Operate on the array before it is written to storage, note that Solidity does not allow `pop()` operations on memory arrays. Therefore, additional modifications, such as tracking the length of the array, are necessary if you wish to implement changes based on the POC above.

Client Response

client response for ethprinter: Fixed. commit-aa157d054df1100b1ce56eee2e1ea58db5c0d3c7

LU2-4:Code optimization

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Bryce

Code Reference

- code/contracts/protocol/LagrangeCommittee.sol#L74

```
74: if (_count > 0) return; // already initialized
```

Description

Bryce: The setFirstEpochPeriod function is used to initialize the epoch period. If the epoch period has already been initialized, it should not continue to be executed. The common implementation method in such cases should be to fail the execution if it has already been initialized, rather than returning and continuing the execution.

Recommendation

Bryce: It is suggested to use require to check if the initialization has already been done. That is, use require(_count == 0) instead of if (_count > 0) return;

Client Response

client response for Bryce: Fixed. commit-aa157d054df1100b1ce56eee2e1ea58db5c0d3c7

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.