Competitive Security Assessment

Lagrange_Update_3

Jul 30th, 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.

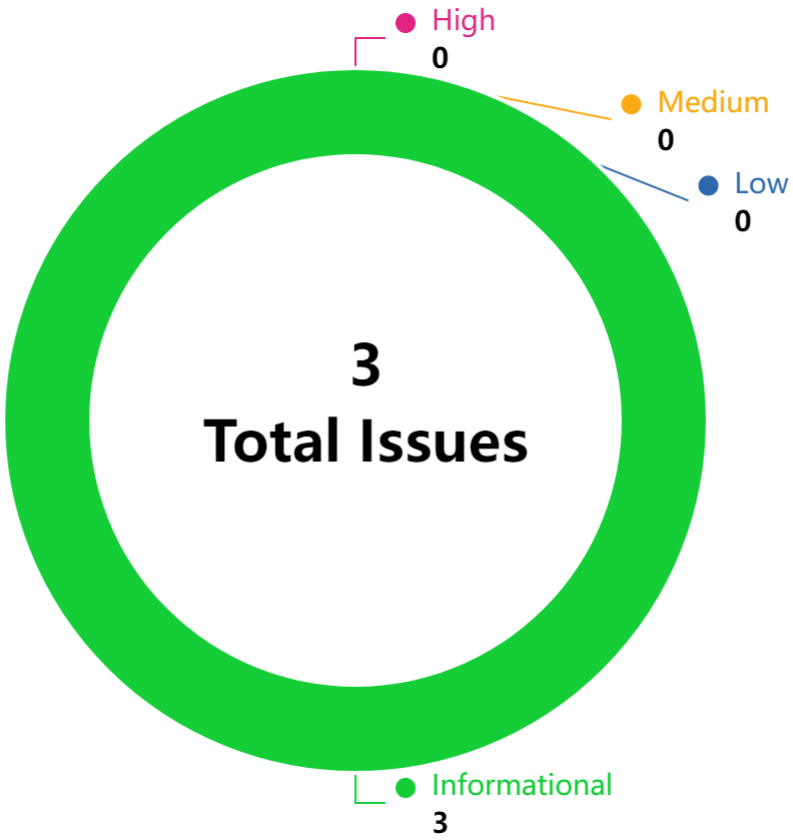
Overview

Project Name	Lagrange_Update_3
Language	solidity
Codebase	• https://github.com/Lagrange-Labs/lsc-contracts • audit version-aa29e6808d7fff60e7dd184b0554ff1ddc7013cb • final version-9d50439a0ba76a7b238a39a4562e9e0a9a527867

Audit Scope

File	SHA256 Hash
contracts/protocol/LagrangeCommittee.sol	f9a7c6c747d5ed51283d44395bc0e6e612ad63f7a334355a5cd0b76f75e8fe26
contracts/protocol/LagrangeService.sol	e5efd67bf61f6bac2df9cb38c9fc0e5a71cd9334f013316044c06b810f12ce83
contracts/library/BLSKeyChecker.sol	8b04d19677793054b64e93e7318a828ea2e239c7c7308b797cec0a6bfe2c06ad

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LA3-1	`BLSKeyChecker` is not compatible with `ERC-7201`	Logical	Informational	Fixed	***
LA3-2	Wrong `DOMAIN_TYPEHASH` definition	Logical	Informational	Fixed	***
LA3-3	The `_orgLength` in `_registerOperator` function is wrong	Logical	Informational	Fixed	***

LA3-1: `BLSKeyChecker` is not compatible with `ERC-7201`

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/library/BLSKeyChecker.sol#L8-L24

```
8: abstract contract BLSKeyChecker is IBLSKeyChecker {
9:     using BN254 for BN254.G1Point;
10:
11:     uint256 internal constant PAIRING_EQUALITY_CHECK_GAS = 120000;
12:
13:     bytes32 public constant DOMAIN_TYPEHASH =
14:         keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
15:
16:     bytes32 public constant BLS_KEY_WITH_PROOF_TYPEHASH =
17:         keccak256("BLSKeyWithProof(address operator,bytes32 salt,uint256 expiry)");
18:
19:     struct SaltStorage {
20:         mapping(address => mapping(bytes32 => bool)) operatorSalts;
21:     }
22:
23:     // keccak256(abi.encode(uint256(keccak256("lagrange.blskeychecker.storage")) - 1)) & ~bytes32(uint256
(0xff))
24:     bytes32 private constant SaltStorageLocation = 0x51615ea63289f14fdd891b383e2929b2f73c675cf292e602b5fce
b059f7a4700;
```

Description

***: ERC-7201: Namespaced Storage Layout is an convention that can be used to avoid storage layout errors when modifying base contracts or when changing the inheritance order of contracts. This convention is used in the upgradeable variant of OpenZeppelin Contracts starting with version 5.0.
The contract `BLSKeyChecker` is using this layout in the following code:

```
struct SaltStorage {
        mapping(address => mapping(bytes32 => bool)) operatorSalts;
    }


    // keccak256(abi.encode(uint256(keccak256("lagrange.blskeychecker.storage")) - 1)) & ~bytes32(uint256(0x
ff))
    bytes32 private constant SaltStorageLocation = 0x51615ea63289f14fdd891b383e2929b2f73c675cf292e602b5fceb0
59f7a4700;
```

As per the ERC-7201: Namespaced Storage Layout, A namespace in a contract should be implemented as a struct type. These structs should be **annotated** with the NatSpec tag `@custom:storage-location <FORMULA_ID>:<NAMESPACE_ID>`, where `<FORMULA_ID>` identifies a formula used to compute the storage location where the namespace is rooted, based on the namespace id. For the struct `SaltStorage`, the NatSpec tag should be `/// @custom:storage-location erc7201:lagrange.blskeychecker.storage`. The issue here is that the contract `BLSKeyChecker` does not define this NatSpec tag for the struct `SaltStorage`.

What's more, the Solidity compiler includes this annotation in the AST since **v0.8.20**, so this is recommended as the minimum compiler version when using this pattern.

Recommendation

***: Consider following fix:

```
pragma solidity 0.8.20;
```

```
/// @custom:storage-location erc7201:lagrange.blskeychecker.storage
struct SaltStorage {
        mapping(address => mapping(bytes32 => bool)) operatorSalts;
    }

    // keccak256(abi.encode(uint256(keccak256("lagrange.blskeychecker.storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant SaltStorageLocation = 0x51615ea63289f14fdd891b383e2929b2f73c675cf292e602b5fceb059f7a4700;
```

Client Response

client response : Fixed in commit 9d50439a0ba76a7b238a39a4562e9e0a9a527867
Secure3: Fixed in commit 9d50439a0ba76a7b238a39a4562e9e0a9a527867

LA3-2:Wrong `DOMAIN_TYPEHASH` definition

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/library/BLSKeyChecker.sol#L13-L14

```
13: bytes32 public constant DOMAIN_TYPEHASH =
14:         keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
```

Description

***: In the build of the DOMAIN TYPEHASH the string version is forgotten.If the project party needs to switch between different versions, then some security issues may arise.
You can find relevant information in the following link.
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md#definition-of-domainseparator
string version :the current major version of the signing domain. Signatures from different versions are not compatible.
https://eips.ethereum.org/EIPS/eip-712#rationale-for-typehash
***: As per EIP712, the definition of `domainSeparator` should include `version`:

```
where the type of eip712Domain is a struct named EIP712Domain with one or more of the below fields. Protocol designers only need to include the fields that make sense for their signing domain. Unused fields are left out of the struct type.

1. string name: the user readable name of signing domain, i.e. the name of the DApp or the protocol.
2. string version: the current major version of the signing domain. Signatures from different versions are not compatible.
3. uint256 chainId: the EIP-155 chain id. The user-agent should refuse signing if it does not match the currently active chain.
4. address verifyingContract: the address of the contract that will verify the signature. The user-agent may do contract specific phishing prevention.
5. bytes32 salt an disambiguating salt for the protocol. This can be used as a domain separator of last resort.
```

However, in contract `BLSKeyChecker` the `DOMAIN_TYPEHASH` lacks of `version`, which break `EIP712`:

```
bytes32 public constant DOMAIN_TYPEHASH =
        keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
```

This may lead to unexpected side-effects.
Note:
You can see the correct definition of `DOMAIN_TYPEHASH` in openzeppelin's EIP712 contract

Recommendation

***: Add the "string version" .

***: Consider following fix:

```
bytes32 public constant DOMAIN_TYPEHASH =
        keccak256("EIP712Domain(string name,string version, uint256 chainId,address verifyingContract)");

function domainSeparator() public view returns (bytes32) {
        return
            keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256("Lagrange State Committee"),"1", block.chainid,
address(this)));
    }
```

Client Response

client response : Fixed in commit 9d50439a0ba76a7b238a39a4562e9e0a9a527867
client response : Fixed in commit 9d50439a0ba76a7b238a39a4562e9e0a9a527867

LA3-3:The `_orgLength` in `_registerOperator` function is wrong

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/protocol/LagrangeCommittee.sol#L428-L443

```
428: function _registerOperator(address _operator, address _signAddress, BLSKeyWithProof memory _blsKeyWithPro
of)
429:         internal
430:     {
431:         _validateBLSKeyWithProof(_operator, _blsKeyWithProof);
432:
433:         delete operatorsStatus[_operator];
434:         OperatorStatus storage _opStatus = operatorsStatus[_operator];
435:         uint256 _orgLength = _opStatus.blsPubKeys.length;
436:         _opStatus.signAddress = _signAddress;
437:         uint256 _length = _blsKeyWithProof.blsG1PublicKeys.length;
438:         for (uint256 i; i < _length; i++) {
439:             _checkBlsPubKeyDuplicate(_opStatus.blsPubKeys, _blsKeyWithProof.blsG1PublicKeys[i]);
440:             _opStatus.blsPubKeys.push(_blsKeyWithProof.blsG1PublicKeys[i]);
441:         }
442:         emit BlsKeyUpdated(_operator, _orgLength, _length, 0);
443:     }
```

Description

***: The `_registerOperator` function will get the length of `blsPubKeys` after deleting `operatorsStatus[_operator]` :

```
delete operatorsStatus[_operator];
OperatorStatus storage _opStatus = operatorsStatus[_operator];
uint256 _orgLength = _opStatus.blsPubKeys.length;
```

Since the `delete` operation will reset all data in `OperatorStatus`, the `_orgLength` will be 0.
The `_orgLength` is used in event `BlsKeyUpdated` :

```
emit BlsKeyUpdated(_operator, _orgLength, _length, 0);
```

As per the definition of `BlsKeyUpdated`, the `_orgLength` is the length of the old `blsPubKeys` in `OperatorStatus`.
However, the `_orgLength` now is always 0, that means the data in event `BlsKeyUpdated` is always wrong.

Recommendation

***: Consider getting `_orgLength` before deleting `operatorsStatus[_operator]` :

```
uint256 _orgLength = operatorsStatus[_operator].blsPubKeys.length;
delete operatorsStatus[_operator];
OperatorStatus storage _opStatus = operatorsStatus[_operator];
```

Client Response

client response : Fixed in commit 9d50439a0ba76a7b238a39a4562e9e0a9a527867

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.