SECURITY AUDIT OF ROSEON TOKEN SMART CONTRACTS

PUBLIC REPORT

MAY 14, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ACRONYMS AND ABBREVIATIONS

NAME	DESCRIPTION
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
Roseon Token (ROSN)	The utility token of Roseon.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on May 14, 2021. We would like to thank Roseon for trusting Verichains Lab in audit smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Roseon Token smart contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

The assessment identified some issues in Roseon smart contract code.

Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Roseon and Roseon Token

Roseon Finance is a multichain mobile first yield aggregator + NFT. Through the easy-to-use Roseon mobile application and web interface, Roseon simplifies DeFi and brings DeFi opportunities. Other features include Swap and NFTs portal access to anyone with a mobile phone and a browser allowing users to manage their crypto and digital tokens from wherever they go.

Roseon Token (ROSN) is the utility token of Roseon, with a total supply of 100 million tokens.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts.

INITIAL VERSION

The initial source code of Roseon smart contracts was provided to Verichains Lab on 26th April 2021, in **roseon_smart_contract.zip** file. All the contract files and checksums are listed in Table 1.

Filename	SHA256
roseon_smart_contract.zip	4bd621a77972a435857959a9b5a7ea1650122b4ffbdbbf8a60f4b483c209ae2b
BEP20.sol	c5da5bed013965b39c3c60a78de530d8320e34442ec60dc76d6d82f7aa584297
IBEP20.sol	88b07ba01a6b601715fe45b97b1fbc6e7ac299f537e32abe43d6399b1d51daf7
LGEWhitelisted.sol	0c4dbec899e6c4f937a0c32bca009f3bfdffbd43ed2afcb5fc77c3a35b5576e1
Roseon.sol	4ee98358d60d2a15eff4afb99f3c7ae00345f91fad3272d90284d89844320438
access/Ownable.sol	57a5c2d1dcbffd00a31b00707b70ff90b71ab497a429f84e4e52c86dcb7d34d5
extensions/IBEP20Metadata.sol	0229b68423bf444d746e07af143cee92a76ee2ef79a7582b7a09f0c841a8dc88
security/Pausable.sol	4f3cc6188b3ead12d05e8151f684ae35154d4507b43ccfbdbf6cd8356de4c52e
utils/Context.sol	543c46d0f81fd4e5d9d6a92beef3d2be18badb483b0b4718c819fe3dbbc37587
utils/SafeMath.sol	6f78c3ed573a960297585ff30b926477aa2ac81297ecae48727de5102c9f7742

Table 1: List of audited files and checksums

UPDATED VERSION

After the first version of this report, the Roseon team have updated the smart contracts and provided its new source codes to Verichains Lab on 9th May 2021, in **roseon_smart_contract_updated_v4.zip** file. All the changed files are listed in Table 2.

Filename	SHA256
roseon_smart_contract_updated_v4.zip	0e8e30b1d6ddf56ef29bc77ef7d16951de711883e720d9a9f2e19314136a1985
BEP20.sol	29af26ec90fac71dada11b1482f9ba52b2ca058812543b6ab385ae5927dd5cb5
IBEP20.sol	bde12c195a4237cb0a4d7960b5ba3fd0585ad83081d8cc559b34b23991c59c9c
LGEWhitelisted.sol	1df8fe3373d4b9c810e3049ca7557b7f88ee4c8768e606e1be912170ccfcc0c3
Roseon.sol	0bdc03a5b8101f9f8045ec594d46dfd2b3c405f3a79f405dd659fbe38051a2b6
access/Ownable.sol	Removed
extensions/IBEP20Metadata.sol	e22ae88a79f94e239be7c1199c4146d87e01925b88d830b238c66c3d814d4534
security/Pausable.sol	e3f17b07cac629101d01c713aed1ea2e51425b0104646e266c965c5f0e7416e4
utils/Context.sol	543c46d0f81fd4e5d9d6a92beef3d2be18badb483b0b4718c819fe3dbbc37587
utils/SafeMath.sol	Removed

Table 2: List of changed files and new checksums

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- TimeStamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories, depending on their severity level:

Severity	Description
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 3: Vulnerability severity levels

1.4. Result summary

Table 4 shows the discovered vulnerabilities and fixing status.

#	Summary	Severity	Status	Update
1	Missing validations for unlock timeline that can make an address locked forever	CRITICAL	☑ Fixed	May 10, 2021
2	Address parameters of events should be indexed	LOW	☑ Fixed	May 10, 2021
3	Hard-coded token name and symbol in super contract LockCoin	LOW	☑ Fixed	May 10, 2021
4	Missing error messages when using require function	LOW	☑ Fixed	May 10, 2021
5	Unnecessary use of SafeMath library	LOW	☑ Fixed	May 10, 2021
6	Typos	LOW	☑ Fixed	May 10, 2021

Table 4. Vulnerabilities summary

1.5. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. DISCOVERED VULNERABILITIES

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.1. Missing validations for unlock timeline that can make an address locked forever [CRITICAL] [FIXED]

When adding a new locking schedule for an address in the `addScheduleLockByAddress` function, the contract does not have any validations for `unlockTime`. But in the while loop condition in `_unlock` function, the contract requires `unlockTime` to be greater than `0` (at line 59 in **Roseon.sol**). So if the unlock timeline has a milestone with `unlockTime = 0`, then the unlock process will always stop at this milestone, or in other words, the remaining locked balance will be locked forever.

Additionally, that invalid milestone's `unlockTime` cannot be updated because the `updateScheduleLock` function requires the current `unlockTime` to be greater than `block.timestamp`, which is always false in this case.

RECOMMENDED FIXES

The contract must have validations for locking timeline (especially for the unlock times) when adding new locking schedule. The valid unlock times (store at `_arrUnlockTime`) should satisfy all of these conditions:

- `_arrUnlockTime[i] > 0`
  for each `0 <= i < arrUnlockTime.length`
- `_arrUnlockTime[i - 1] < _arrUnlockTime[i]`
  for each `1 <= i < arrUnlocktime.length`

UPDATES

The updated smart contracts have added the following modifications to fix this issue:

- Adding check for valid unlock timeline when adding/updating locking schedule.
- Removing check for `unlockTime > 0` in `_unlock` function.

2.2. Address parameters of events should be indexed [LOW] [FIXED]

The `Unlock` and `AddAddressLock` events should have `indexed` keywords for `addressLock` parameters. This would help us filter events related to a specific address easily.

Here is the current signature of these two events in the **LockCoin** contract:

contracts/Roseon.sol#L15:16

```
event Unlock(address addressLock, uint256 amount);
event AddAddressLock(address addressLock, uint256 amount);
```

RECOMMENED FIXES

Add *indexed* keyword to the *addressLock* parameters in these events.

contracts/Roseon.sol#L15:16

```
event Unlock(address indexed addressLock , uint256 amount);
event AddAddressLock(address indexed addressLock, uint256 amount);
```

UPDATES

The updated smart contracts have added the indexed keyword to these events.

2.3. Hard-coded token name and symbol in super contract LockCoin [LOW] [FIXED]

Name and symbol of **Roseon** token should be specified in the derived contract (**RoseonToken**) instead of hard-coded in the super contract (**LockCoin**).

contracts/Roseon.sol

```
contract LockCoin is Ownable, BEP20 {
    using SafeMath for uint256;

    constructor() BEP20("Roseon token", "ROSN") {
        _mint(msg.sender, 100000000 * 10**18);
    }

    /** ... */
}
```

RECOMMENED FIXES

We should change the constructors of **LockCoin** (super contract) and **RoseonToken** (derived contract) as below:

contracts/Roseon.sol#L51:L73

```
contract LockCoin is Ownable, BEP20 {
    using SafeMath for uint256;

    constructor(string memory name, string memory symbol) BEP20(name, symbol) {
        _mint(msg.sender, 100000000 * 10**18);
    }

    /** ... */
}

contract RoseonToken is LockCoin, LGEWhitelisted {
    using SafeMath for uint256;

    constructor() LockCoin("Roseon token", "ROSN") {}

    /** ... */
}
```

UPDATES

The updated smart contracts have moved the name, symbol and total supply initialization to the **RoseonToken** constructor.

2.4. Missing error messages when using require function [LOW] [FIXED]

The *transfer* and *transferFrom* functions inside the *RoseonToken* contract are using *require* function without an error message.

contracts/Roseon.sol#L154:168

```
function transfer(address _receiver, uint256 _amount) public override returns (bool success) {
    /** ... */
    return BEP20.transfer(_receiver, _amount);
}

function transferFrom(
    address _from,
    address _receiver,
    uint256 _amount
) public override returns (bool) {
    /** ... */
    return BEP20.transferFrom(_from, _receiver, _amount);
}
```

RECOMMENDED FIXES

The *require* functions should be used with error messages, for example:

contracts/Roseon.sol#L154:168

```
function transfer(address _receiver, uint256 _amount) public override returns (bool success) {
    /** ... */
    return BEP20.transfer(_receiver, _amount, "Balance is insufficient");
}

function transferFrom(
    address _from,
    address _receiver,
    uint256 _amount
) public override returns (bool) {
    /** ... */
    return BEP20.transferFrom(_from, _receiver, _amount, "Balance is insufficient");
}
```

UPDATES

The updated smart contracts have added error messages to the *require* functions as recommended.

2.5. Unnecessary use of SafeMath library [LOW] [FIXED]

Starting from **Solidity** version 0.8.0[1], the compiler has built in overflow checking, so all arithmetic operations will be reverted on overflow exceptions. So basically, we don't need to import and use the Openzeppelin's **SafeMath** library[2] anymore.

UPDATES

The updated smart contracts have removed the **SafeMath** library as recommended.

2.6. Typos [LOW] [FIXED]

There are some typos in the current source code of **Roseon** smart contracts:

- File: `Roseon.sol`
  - Line 49: @dev Unlock token of "_addressLock" with ~~time line~~timeline lock
  - Line 95: "Cannot change ~~time line~~timeline in the past"
  - Line 114: @dev get lock time and amoutamount lock by address at a time
  - Line 124: @dev add list ~~time line~~timeline lock and total amount lock by address

UPDATES

The updated smart contracts have fixed all these typos.

[1] https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html#silent-changes-of-the-semantics
[2] https://docs.openzeppelin.com/contracts/4.x/utilities#api:math.adoc#SafeMath

3. VERSION HISTORY

Version	Date	Status/Changes	Created by
1.0	May 06, 2021	Initial report	Verichains Lab
2.0	May 10, 2021	Updated report	Verichains Lab
2.1	May 14, 2021	Public report	Verichains Lab