Smart Contract Security Audit Report

The SlowMist Security Team received the Avalon Financa team's application for smart contract security audit of the Avalon Governance Token on 2024.12.24. The following are the details and results of this smart contract security audit:

Token Name :

Avalon Governance Token

The contract address :

https://github.com/avalonfinancexyz/avalon-governance-token

commit: 286bcdac8ee635d3c7efc105ac4a5d821702907b

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed

NO.	Audit Items	Result
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002412260001

Audit Date : 2024.12.24 - 2024.12.26

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the token vault section and the dark coin functions. The total amount of contract tokens can be changed. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. The ADMIN_ROLE can grant the MANAGER_ROLE, the MANAGER_ROLE can add and remove the users from the blacklist. And the blacklist address can not transfer tokens.
2. The ADMIN_ROLE can grant the BURN_ROLE, and the BURN_ROLE can burn the blacklist user's tokens through the burn function.
3. The ADMIN_ROLE can grant the MINT_ROLE, the MINT_ROLE can mint tokens through the mint function and there is an upper limit on the amount of tokens.

After communication with the project team, they stated that the `_delegate` address would be a multi-sig contract address, and the admin and minter roles would use a timelock to control.

The source code:

Avalon.sol

```
// SPDX-License-Identifier: UNLICENSED
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.22;

import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol";
import {OwnableUpgradeable} from "@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol";
import {OFTUpgradeable} from "@layerzerolabs/oft-evm-
```

```
upgradeable/contracts/oft/OFTUpgradeable.sol";

contract Avalon is OFTUpgradeable, AccessControlUpgradeable, PausableUpgradeable {
    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
    bytes32 public constant BURN_ROLE = keccak256("BURN_ROLE");
    bytes32 public constant PAUSE_ROLE = keccak256("PAUSE_ROLE");

    mapping(address => bool) public isBlackListed;

    event AddedBlackList(address _addr);
    event RemovedBlackList(address _addr);

    error BlackListed();
    error NotInBlackList();

    constructor(address _lzEndpoint) OFTUpgradeable(_lzEndpoint) {
        _disableInitializers();
    }

    function initialize(string memory _name, string memory _symbol, address
_delegate) public initializer {
        __OFT_init(_name, _symbol, _delegate);
        __Ownable_init(_delegate);

        _grantRole(DEFAULT_ADMIN_ROLE, _delegate);
        _grantRole(ADMIN_ROLE, _delegate);
        _setRoleAdmin(MANAGER_ROLE, ADMIN_ROLE);
        _setRoleAdmin(BURN_ROLE, ADMIN_ROLE);
        _setRoleAdmin(PAUSE_ROLE, ADMIN_ROLE);
        _setRoleAdmin(ADMIN_ROLE, ADMIN_ROLE);
    }

    function addBlackList(address _addr) public onlyRole(MANAGER_ROLE) {
        isBlackListed[_addr] = true;
        emit AddedBlackList(_addr);
    }

    function removeBlackList(address _addr) public onlyRole(MANAGER_ROLE) {
        isBlackListed[_addr] = false;
        emit RemovedBlackList(_addr);
    }
    //SlowMist// Suspending all transactions upon major abnormalities is a recommended
approach
    function pause() external onlyRole(PAUSE_ROLE) {
        PausableUpgradeable._pause();
    }

    function unpause() external onlyRole(PAUSE_ROLE) {
```

```
        PausableUpgradeable._unpause();
    }

    function burn(address _user, uint256 _amount) public virtual onlyRole(BURN_ROLE)
{
        if (!isBlackListed[_user]) revert NotInBlackList();
        _burn(_user, _amount);
    }

    function _update(address from, address to, uint256 value) internal virtual
override whenNotPaused {
        if ((isBlackListed[from] && to != address(0)) || isBlackListed[to]) revert
BlackListed();
        super._update(from, to, value);
    }
}
```

AvalonMintable.sol

```
// SPDX-License-Identifier: UNLICENSED
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.22;

import "@openzeppelin/contracts-
upgradeable/token/ERC20/extensions/ERC20CappedUpgradeable.sol";

import "./Avalon.sol";

contract AvalonMintable is Avalon, ERC20CappedUpgradeable {
    bytes32 public constant MINT_ROLE = keccak256("MINT_ROLE");

    uint256 public mintedAmount;

    event MintedAmountUpdated(uint256 amount);

    constructor(address _lzEndpoint) Avalon(_lzEndpoint) {
        _disableInitializers();
    }

    function initialize(string memory _name, string memory _symbol, address
_delegate, uint256 _cap)
        public
        initializer
    {
        __ERC20Capped_init(_cap);
        super.initialize(_name, _symbol, _delegate);
```

```
        _setRoleAdmin(MINT_ROLE, ADMIN_ROLE);
    }
    //SlowMist// The MINT_ROLE can mint tokens through the mint function and there is
an upper limit on the amount of tokens.
    function mint(uint256 _amount) public virtual onlyRole(MINT_ROLE) {
        _mint(owner(), _amount);

        mintedAmount += _amount;
        emit MintedAmountUpdated(mintedAmount);

        uint256 maxSupply = cap();
        if (mintedAmount > maxSupply) revert ERC20ExceededCap(mintedAmount,
maxSupply);
    }

    function _update(address from, address to, uint256 value) internal
override(ERC20CappedUpgradeable, Avalon) {
        super._update(from, to, value);
    }
}
```

AvalonOFTAdapter.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.22;

import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {OFTAdapterUpgradeable} from "@layerzerolabs/oft-evm-
upgradeable/contracts/oft/OFTAdapterUpgradeable.sol";

/**
 * @title OFTAdapter Contract
 * @dev OFTAdapter is a contract that adapts an ERC-20 token to the OFT
functionality.
 *
 * @dev For existing ERC20 tokens, this can be used to convert the token to
crosschain compatibility.
 * @dev WARNING: ONLY 1 of these should exist for a given global mesh,
 * unless you make a NON-default implementation of OFT and needs to be done very
carefully.
 * @dev WARNING: The default OFTAdapter implementation assumes LOSSLESS transfers,
ie. 1 token in, 1 token out.
 * IF the 'innerToken' applies something like a transfer fee, the default will NOT
work...
 * a pre/post balance check will need to be done to calculate the
amountSentLD/amountReceivedLD.
 */
```

```
contract AvalonOFTAdapter is OFTAdapterUpgradeable {
    constructor(address _token, address _lzEndpoint) OFTAdapterUpgradeable(_token,
_lzEndpoint) {
        _disableInitializers();
    }

    function initialize(address _delegate) public initializer {
        __OFTAdapter_init(_delegate);
        __Ownable_init(_delegate);
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist