SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.28, the SlowMist security team received the FRENS team's security audit application for frens-contracts-v2, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This project is an ETH pledge agreement, including a stake pool contract and an ERC721 token contract.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Redundant code	Others	Suggestion	Acknowledged
N2	The calling function is not implemented	Others	Low	Acknowledged

NO	Title	Category	Level	Status
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged
N4	Return value not checked	Others	Low	Fixed
N5	Function addToDeposit does not check msg.value	Others	Suggestion	Fixed
N6	The function withdraw does not check the _id parameter	Others	Suggestion	Fixed
N7	Missing event record	Others	Suggestion	Fixed
N8	Redundant code	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/frens-pool/frens-contracts-v2/tree/main

Initial audit commit: c1cfa72833e6811c48d45502877a078df41d5dc3

Final audit commit: 2435be6d21f82950d7adfb2bdf13e4f520c38610

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

StakingPool			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify	initializer

StakingPool			
		State	
depositToPool	External	Payable	noZeroValueTxn mustBeAccepting maxTotDep
addToDeposit	External	Payable	mustBeAccepting maxTotDep correctPoolOnly
stake	External	Can Modify State	onlyOwner
stake	External	Can Modify State	onlyOwner
_stake	Internal	Can Modify State	-
setPubKey	External	Can Modify State	onlyOwner
_setPubKey	Internal	Can Modify State	-
withdraw	External	Can Modify State	mustBeAccepting
claim	External	Can Modify State	correctPoolOnly
getIdsInThisPool	External	-	-
getShare	External	-	correctPoolOnly
_getShare	Internal	-	-
getDistributableShare	External	-	-
getState	External	-	-
owner	Public	-	-
_toWithdrawalCred	Private	-	-
setArt	External	Can Modify State	onlyOwner
callSSVNetwork	External	Can Modify State	onlyOwner
transferToken	External	Can Modify State	onlyOwner

StakingPool			
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

FrensPoolShare			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
mint	Public	Can Modify State	-
exists	Public	-	-
getPoolById	Public	-	-
getApproved	Public	-	-
tokenURI	Public	-	-
renderTokenById	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-
burn	Public	Can Modify State	-
supportsInterface	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant code

Category: Others

Content

1.The FrensPoolShare contract imports openzeppelin's Ownable module, but it is not used in the contract.

FrensPoolShare.sol#L14

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

2.In the StakingPool contract, the `currentState` variable is always the value in enum `PoolState`, because when the StakingPoolFactory contract creates the StakingPool contract, it also calls the `initialize` function to initialize the value of currentState.Therefore it is impossible for the `getState` function to return `"state failure"`.

StakingPool.sol#L365-L373

```
function getState() external view returns (string memory) {
    if (currentState == PoolState.awaitingValidatorInfo)
        return "awaiting validator info";
    if (currentState == PoolState.staked) return "staked";
    if (currentState == PoolState.acceptingDeposits)
        return "accepting deposits";
    if (currentState == PoolState.exited) return "exited";
    return "state failure"; //should never happen
}
```

3.In the FrensPoolShare function, the `burn` function can only be called by the StakingPool contract, but the use of the burn function is not implemented in the StakingPool contract.

FrensPoolShare,sol#L96-L102

```
function burn(uint tokenId) public {
    require(
        msg.sender == address(poolByIds[tokenId]),
        "cannot burn shares from other pools"
    );
    _burn(tokenId);
}
```

4.In the StakingPool contract, some variables are not used in this audited version.

StakingPool.sol

```
struct RageQuit {
    uint price;
    uint time;
    bool rageQuitting;
}
mapping(uint => bool) public locked;
mapping(uint => RageQuit) public rageQuitInfo;
bool public transferLocked;
```

Solution

It is recommended to remove redundant code.

Status

Acknowledged; The project team explains this as follows:

1. Ownable is used here because it allows the contract owner to modify the landing page on OpenSea. It is not used anywhere in the contract itself, but they want control over the way the collection is displayed on opensea, and making the contract ownable is an easy way to do that.
2. “state failure” should never occur - it is included for syntactic sugar, and we will leave this as-is.
3. The burn function in the PoolShare contract is present to accommodate future functionality. It is included so that the PoolShare contract can be non-upgradeable, and still accommodate this planned upgrade in the future.
4. RageQuit is part of a planned future upgrade, and needs to be included in the StakingPool contract, as it is accessed by the PoolShare contract.

[N2] [Low] The calling function is not implemented

Category: Others

Content

In the FrensPoolShare contract, the `getApproved` function calls the `rageQuitInfo` function of the stakingPool contract, but this function is not implemented.

FrensPoolShare.sol#L57-L68

```
    function getApproved(uint256 tokenId) public view virtual override(ERC721,
IERC721) returns (address) {
        _requireMinted(tokenId);
        address poolAddr = poolByIds[tokenId];
        IStakingPool stakingPool = IStakingPool(poolAddr);
        (/*price*/,/*time*/, bool quitting) = stakingPool.rageQuitInfo(tokenId);
        if(quitting) {
            return poolAddr;
        } else{
            return super.getApproved(tokenId);
        }
}
```

Solution

It is recommended to implement the rageQuitInfo function in stakingPool or delete the getApproved function.

Status

Acknowledged; The project team stated that this part will be implemented in future upgrades.

[N3] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the FrensPoolShare contract, `DEFAULT_ADMIN_ROLE` can add any address as `MINTER_ROLE` through the AccessControl module of openzeppelin imported into the contract. `MINTER_ROLE` can mint FRENS Share (ERC721 token) through the `mint` function, and can also burn the FRENS Share minted by it through the `burn` function.

FrensPoolShare.sol#L13,L38-L46,L96-L102

```
import "@openzeppelin/contracts/access/AccessControl.sol";

function mint(address userAddress) public {
    require(
        hasRole(MINTER_ROLE, msg.sender),
        "you are not allowed to mint"
    );
    uint256 _id = totalSupply();
    poolByIds[_id] = address(msg.sender);
    _safeMint(userAddress, _id);
}

 function burn(uint tokenId) public {
    require(
        msg.sender == address(poolByIds[tokenId]),
        "cannot burn shares from other pools"
    );
    _burn(tokenId);
}
```

2.In the StakingPool contract, the owner role can perform staking operations through two `stake` functions with the same name; the pool validator information can be set through the `setPubKey` function; the artForPool address can be set through the `setArt` function; data can be transferred to the ssvNetwork contract through the `callSSVNetwork` function; and `transferToken` function can be used to transfer ERC20 tokens in the contract.Openzeppelin's OwnableUpgradeable module is imported into the contract, and the `owner` role can transfer its ownership to other addresses.

StakingPool,sol#L8,L193-L212,L215-L217,L235-L247,L390-L395,L397-L401,L403-L406

```
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

function stake(
    bytes calldata _pubKey,
    bytes calldata _withdrawal_credentials,
    bytes calldata _signature,
    bytes32 _deposit_data_root
) external onlyOwner {
    //if validator info has previously been entered, check that it is the same,
then stake
    if (validatorSet) {
        require(keccak256(_pubKey) == keccak256(pubKey), "pubKey mismatch");
    } else {
        //if validator info has not previously been entered, enter it, then stake
        _setPubKey(
            _pubKey,
            _withdrawal_credentials,
            _signature,
            _deposit_data_root
        );
    }
    _stake();
}

function stake() external onlyOwner {
    _stake();
}

function setPubKey(
    bytes calldata _pubKey,
    bytes calldata _withdrawal_credentials,
    bytes calldata _signature,
    bytes32 _deposit_data_root
) external onlyOwner {
    _setPubKey(
        _pubKey,
        _withdrawal_credentials,
        _signature,
            _deposit_data_root
        );
    }

    function setArt(IFrensArt newArtContract) external onlyOwner {
        IFrensArt newFrensArt = newArtContract;
        string memory newArt = newFrensArt.renderTokenById(0);
        require(bytes(newArt).length != 0, "invalid art contract");
        artForPool = newArtContract;
    }

    function callSSVNetwork(bytes memory data) external onlyOwner {
        address ssvNetwork =
frensStorage.getAddress(keccak256(abi.encodePacked("external.contract.address",
"SSVNetwork")));
        (bool success, ) = ssvNetwork.call(data);
        require(success, "Call failed");
    }

        function transferToken(address tokenAddress, address to, uint amount)
external onlyOwner {
        IERC20 token = IERC20(tokenAddress);
        token.transfer(to, amount);
    }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged; After verification, the DEFAULT_ADMIN_ROLE role address of the FrensPoolShare contract is the multisig wallet contract.In the future the intention is to have this be community managed. Since each pool instance is separately managed by its owner, each pool can decide if it wants to have that be an EOA or a Multisig.

[N4] [Low] Return value not checked

Category: Others

Content

In the StakingPool contract, when the `initialize` function calls the `approve` function of the `ssvTokenAddress` token contract and the `transferToken` function calls the `transfer` of the `tokenAddress` token contract, the return value of the function is not checked.

StakingPool.sol#L126-L153,L403-L406

~~~
    function initialize(
        address owner_,
        bool validatorLocked_,
        IFrensStorage frensStorage_
    ) public initializer() {
    ```      IERC20(ssvTokenAddress).approve(ssvNetwork,type(uint256).max);
    ```
    }

    function transferToken(address tokenAddress, address to, uint amount) external
onlyOwner {
        IERC20 token = IERC20(tokenAddress);
        token.transfer(to, amount);
    }
~~~

Solution

It is recommended to check the return value of the function.

Status

Fixed; The project team stated that ERC20 transfer return value check will be included in the next contract version.

[N5] [Suggestion] Function addToDeposit does not check msg.value

Category: Others

Content

In the StakingPool contract, the `addToDeposit` function does not check the value of `msg.value`, and users can pass in 0 ETH for invalid operations.

StakingPool.sol#L185-L190

```
    function addToDeposit(uint _id) external payable mustBeAccepting maxTotDep
correctPoolOnly(_id){
        require(frensPoolShare.exists(_id), "id does not exist"); //id must exist
        depositForId[_id] += msg.value;
        totalDeposits += msg.value;
```

```
        emit AddToDeposit(_id, msg.value);
    }
```

Solution

It is recommended to add the noZeroValueTxn modifier in the addToDeposit function.

Status

Fixed; The project team stated that noZeroValueTxn modifier will be added to addToDeposit in the next contract version.

[N6] [Suggestion] The function withdraw does not check the _id parameter

Category: Others

Content

In the StakingPool contract, the `withdraw` function does not check whether the parameter `_id` belongs to the pool. Users can pass in an `_id` that does not belong to the pool to perform invalid operations.

StakingPool.sol#L279-L287

```
    function withdraw(uint _id, uint _amount) external mustBeAccepting {
        require(msg.sender == frensPoolShare.ownerOf(_id), "not the owner");
        require(depositForId[_id] >= _amount, "not enough deposited");
        depositForId[_id] -= _amount;
        totalDeposits -= _amount;
        (bool success, /*return data*/) = frensPoolShare.ownerOf(_id).call{value:
_amount}("");
        assert(success);
        emit Withdraw(_id, _amount, msg.sender);
    }
```

Solution

It is recommended to add the correctPoolOnly modifier to the withdraw function.

Status

Fixed; The project team stated that correctPoolOnly will be added for next version.

[N7] [Suggestion] Missing event record

Category: Others

Content

In the StakingPool contract, the `setArt` function lacks event records when setting the `artForPool` contract variable.

StakingPool.sol#L390-L395

```
function setArt(IFrensArt newArtContract) external onlyOwner {
    IFrensArt newFrensArt = newArtContract;
    string memory newArt = newFrensArt.renderTokenById(0);
    require(bytes(newArt).length != 0, "invalid art contract");
    artForPool = newArtContract;
}
```

Solution

It is recommended to add events to the setArt function to record.

Status

Fixed; The project team stated that SetArt event will be included in the next contract version..

[N8] [Suggestion] Redundant code

Category: Others

Content

In the StakingPool contract, the `initialize` function calls the `__Ownableinit` function to set the `owner_` address to the `owner` role, and then calls the `transferOwnership` function to repeatedly transfer the owner role to the `owner_` address.

StakingPool.sol#L126-L162

```
function initialize(
    address owner_,
    bool validatorLocked_,
    IFrensStorage frensStorage_
) public initializer() {
    ```
    transferOwnership(owner);
}
```

**Solution**

It is recommended to remove redundant code.
```

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404300004	SlowMist Security Team	2024.04.28 - 2024.04.30	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 low risk, 5 suggestion.

6 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