Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.04.24, the SlowMist security team received the STRX Protocol team's security audit application for STRX Protocol, 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

Audit Version:

https://github.com/justlend/strx-protocol

commit: 38a60d393145a0202814eb870191db56637f6787

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Coding specification	Others	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N2	Gas optimization	Gas Optimization Audit	Suggestion	Acknowledged
N3	Authority transfer enhancement	Others	Suggestion	Acknowledged
N4	Missing onlyStrx modifier	Authority Control Vulnerability Audit	Suggestion	Acknowledged
N5	resourceType missing check	Design Logic Audit	Suggestion	Acknowledged
N6	Business logic is unclear	Design Logic Audit	Suggestion	Acknowledged
N7	Business logic issue	Design Logic Audit	Suggestion	Acknowledged
N8	Code optimization	Others	Suggestion	Acknowledged
N9	Excessive authority issue	Authority Control Vulnerability Audit	Medium	Acknowledged
N10	init lacks parameter validity checking	Others	Suggestion	Acknowledged
N11	Lack of access control	Authority Control Vulnerability Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

STRXG1: TSe1pcCnU1tLdg69JvbFmQirjKwTbxbPrG

STRXProxy: TU3kjFuhtEo42tsCBtfYUAZxoqQ4yuSLQ5

MarketG1: TNoHbPuBQrVanVf9qxUsSvHdB2eDkeDAKD

MarketProxy: TU2MJ5Veik1LRAgjeSzEdvmDYx7mefJZvd

VoteManager: TBx8avtCuPiuYCQfRq97JLpmdnxr2hWR9L

4.2 Visibility Description

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

AdminProxy			
Function Name	Visibility	Mutability	Modifiers
<Fallback>	External	Payable	-
_setPendingImplementation	Public	Can Modify State	-
_acceptImplementation	Public	Can Modify State	-
_setPendingAdmin	Public	Can Modify State	-
_acceptAdmin	Public	Can Modify State	-

BaseSTRX			
Function Name	Visibility	Mutability	Modifiers
approve	Public	Can Modify State	-
decreaseAllowance	Public	Can Modify State	-
increaseAllowance	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
allowance	Public	-	-
balanceOf	Public	-	-
totalUnderlying	Public	-	-
totalSupply	Public	-	-
exchangeRate	Public	-	-
decimals	External	-	-
name	External	-	-
symbol	External	-	-

BaseSTRX			
_approve	Internal	Can Modify State	-
_doTransferOut	Internal	Can Modify State	-
_transfer	Internal	Can Modify State	-
depositInternal	Internal	Can Modify State	-
updateExchangeRate	Internal	Can Modify State	-
withdrawInternal	Internal	Can Modify State	-
withdrawExactInternal	Internal	Can Modify State	-
updateTokensAndDeposits	Private	Can Modify State	-

STRXG1			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-
init	External	Payable	onlyAdmin
_become	External	Can Modify State	-
deposit	External	Payable	-
withdraw	External	Can Modify State	_updateIndex
withdrawExact	External	Can Modify State	_updateIndex
claim	External	Can Modify State	-
claimAll	External	Can Modify State	-
voteSrs	External	Can Modify State	onlyVoteOperator
receiveRental	External	Payable	onlyMarket

STRXG1			
marketDelegateResource	External	Can Modify State	onlyMarket
marketUndelegateResource	External	Can Modify State	onlyMarket
unfreezeBalance	External	Can Modify State	_updateIndex
freezeBalance	External	Can Modify State	_updateIndex
updateIndex	External	Can Modify State	_updateIndex
setReserveFactors	External	Can Modify State	onlyAdmin _updateIndex
setReserveAdmin	External	Can Modify State	onlyAdmin
setVoteOperator	External	Can Modify State	onlyVoteOperatorOrAdmin
setMinFreezeAmount	External	Can Modify State	onlyVoteOperator
setMaxAmountForEnergy	External	Can Modify State	onlyAdmin
claimReserves	External	Can Modify State	onlyReserveAdmin
reservePayBadDebt	External	Can Modify State	onlyReserveAdmin
balanceOfUnderlying	External	Can Modify State	_updateIndex
balanceStructure	External	Can Modify State	_updateIndex
assetsStructure	External	Can Modify State	_updateIndex
supportsInterface	Public	-	-
viewBalanceOfUnderlying	Public	-	-
viewBalanceStructure	Public	-	-
viewAssetsStructure	Public	-	-

STRXG1			
totalDelegated	Public	-	-
totalUnfreezable	Public	-	-
totalFrozen	Public	-	-
totalFrozenOfType	Public	-	-
getBalanceToUnfreeze	Public	-	-
availableUnfreezeSize	Public	-	-
getUnfreezeDelayDays	Public	-	-
getVoteRewardAmount	Public	-	-
_deposit	Internal	Can Modify State	_updateIndex
_settleIncome	Internal	Can Modify State	-
_addReserves	Internal	Can Modify State	-
_updateWithdrawList	Internal	Can Modify State	-
_withdrawRewardRentalAndUpdateIndex	Internal	Can Modify State	-
_updateClaimableAndTransfer	Internal	Can Modify State	-
_freezeBalance	Private	Can Modify State	-
_unfreezeBalance	Private	Can Modify State	-
_getFreezeAmountForResourceType	Private	-	-
_getUnfreezeResourceType	Private	-	-

STRXV1Storage			
Function Name	Visibility	Mutability	Modifiers

STRXV1Storage			
claimable	Public	-	-
withdrawal	Public	-	-
balanceToUnfreeze	Public	-	-
balanceToFreeze	Public	-	-
setClaimable	Internal	Can Modify State	-
setBalanceToUnfreeze	Internal	Can Modify State	-
setBalanceToFreeze	Internal	Can Modify State	-
setWithdrawal	Internal	Can Modify State	-

JumpRateModel			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
replaceOwner	External	Can Modify State	onlyOwner
renounceOwner	External	Can Modify State	onlyOwner
updateJumpRateModel	External	Can Modify State	onlyOwner
getRentalRate	External	-	-
updateJumpRateModelInternal	Internal	Can Modify State	-
utilizationRate	Internal	-	-

MarketG1			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	onlyAdmin
_become	External	Can Modify State	-

MarketG1			
claimRental	External	Can Modify State	onlyStrx
repayBadDebt	External	Payable	-
rentResource	External	Payable	onlyRentNotPaused onlyBandwidthOrEnergy canRent
returnResource	External	Can Modify State	-
returnResourceByReceiver	External	Can Modify State	-
liquidate	External	Can Modify State	onlyBandwidthOrEnergy
forceLiquidate	External	Can Modify State	onlyBandwidthOrEnergy
badDebtGap	External	-	-
getRentInfo	External	-	onlyBandwidthOrEnergy
supportsResourceType	External	-	-
supportsInterface	Public	-	-
setMaxRentalScaleLimit	Public	Can Modify State	onlyAdmin
setBandwidthRentalRateModel	Public	Can Modify State	onlyAdmin
setEnergyRentalRateModel	Public	Can Modify State	onlyAdmin
setRateModel	Public	Can Modify State	onlyAdmin onlyBandwidthOrEnergy
setMinFee	Public	Can Modify State	onlyAdmin
setFeeRatio	Public	Can Modify State	onlyAdmin
setRentPaused	Public	Can Modify State	onlyAdmin onlyBandwidthOrEnergy
setUsageChargeRatio	Public	Can Modify State	onlyAdmin

MarketG1			
updateRentRate	Public	Can Modify State	-
totalDelegated	Public	-	-
totalDelegatedOfType	Public	-	-
totalUnfreezable	Public	-	-
totalUnfreezableOfType	Public	-	-
totalFrozen	Public	-	-
totalFrozenOfType	Public	-	-
delegatableOfType	Public	-	-
maxRentableOfType	Public	-	-
_liquidateRate	Public	-	-
_stableRate	Public	-	-
_rentalRate	Public	-	-
_returnResource	Internal	Can Modify State	onlyBandwidthOrEnergy canReturn
_internalRepayForLiquidate	Internal	Can Modify State	-
_internalRepayForReturn	Internal	Can Modify State	-
_updateMarketAndSettle	Internal	Can Modify State	-
_updateMarketWithRentUser	Internal	Can Modify State	-
_updateRentIndexAndCalcIncome	Internal	Can Modify State	-
_updateUserRentIndex	Internal	Can Modify State	-
_settleIncome	Internal	Can Modify State	-

MarketG1			
_payBadDebt	Internal	Can Modify State	-
_doTransferOut	Internal	Can Modify State	-
_updateRentRate	Internal	Can Modify State	-
_setRateModel	Internal	Can Modify State	-
_calculateFee	Internal	-	-
_liquidateCheck	Private	-	-

MarketProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

MarketV1Storage			
Function Name	Visibility	Mutability	Modifiers
setRental	Internal	Can Modify State	-
rentals	Public	-	-

TRC165			
Function Name	Visibility	Mutability	Modifiers
supportsInterface	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Coding specification

Category: Others

Content

Functions with visibility for Public and External are named with the beginning of _.

Code location:

AdminProxy._setPendingImplementation

AdminProxy._acceptImplementation

AdminProxy._setPendingAdmin

AdminProxy._acceptAdmin

MarketG1._become

MarketG1._liquidateRate

MarketG1._stableRate

MarketG1._rentalRate

STRXG1._become

Functions of the Internal and Private types do not start with _.

Code location:

BaseSTRX.depositInternal(Internal)

BaseSTRX.updateExchangeRate(Internal)

BaseSTRX.withdrawInternal(Internal)

BaseSTRX.withdrawExactInternal(Internal)

BaseSTRX.updateTokensAndDeposits(Private)

JumpRateModel.updateJumpRateModelInternal(Internal)

JumpRateModel.utilizationRate(Internal)

MarketV1Storage.setRental(Internal)

STRXV1Storage.setClaimable(Internal)

STRXV1Storage.setBalanceToUnfreeze(Internal)

STRXV1Storage.setBalanceToFreeze(Internal)

STRXV1Storage.setWithdrawal(Internal)

Solution

It is recommended that the beginning of _ is used for functions of the internal and private types.

Status

Acknowledged

[N2] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

address(0) is a black hole address, nobody can have an address(0), so it looks like you can judge the msg.sender without having to judge the address(0).

Code location: strx-protocol/src/AdminProxy.sol#L65

```
function _acceptImplementation() public {
        // Check caller is pendingImplementation and pendingImplementation ≠
address(0)
        require(msg.sender == pendingImplementation && pendingImplementation !=
address(0),
                "ACCEPT_PENDING_IMPLEMENTATION_ADDRESS_CHECK");

        // Save current values for inclusion in log
        address oldImplementation = implementation;
        address oldPendingImplementation = pendingImplementation;

        implementation = pendingImplementation;

        pendingImplementation = address(0);

        emit NewImplementation(oldImplementation, implementation);
        emit NewPendingImplementation(oldPendingImplementation,
pendingImplementation);
    }
```

Code location: strx-protocol/src/AdminProxy.sol#L106

```
function _acceptAdmin() public {
        // Check caller is pendingAdmin and pendingAdmin ≠ address(0)
        require(msg.sender == pendingAdmin && pendingAdmin != address(0),
"ACCEPT_ADMIN_PENDING_ADMIN_CHECK");
```

```
        // Save current values for inclusion in log
        address oldAdmin = admin;
        address oldPendingAdmin = pendingAdmin;

        // Store admin with value pendingAdmin
        admin = pendingAdmin;

        // Clear the pending value
        pendingAdmin = address(0);

        emit NewAdmin(oldAdmin, admin);
        emit NewPendingAdmin(oldPendingAdmin, pendingAdmin);
    }
```

Solution

It is recommended to delete unnecessary code to optimize Gas.

Status

Acknowledged; The project team responded that the reason they judge here is to avoid the collision of the private key of the 0x0 address with a very small probability, because the default (empty) pendingAdmin and pendingImplementation are 0x0 addresses.

[N3] [Suggestion] Authority transfer enhancement

Category: Others

Content

The owner does not adopt the pending and access processes. If the owner is incorrectly set, the owner permission will be lost.

Code location: strx-protocol/src/JumpRateModel.sol

```
function replaceOwner(address newOwner) external onlyOwner {
        require(newOwner != address(0), "new owner is zero address");
        address oldOwner = owner;
        owner = newOwner;

        emit OwnerUpdated(oldOwner, newOwner);
    }
```

Solution

It is recommended to adopt the pending and access processes. Only the new owner accepts the permissions to transfer.

Status

Acknowledged; The project team responded that the authority would not be lost because the JumpRateModel address can be modified in the Market, so they designed to replace the owner with the JumpRateModel in one step to simplify the operation.No adjustment was needed.

[N4] [Suggestion] Missing onlyStrx modifier

Category: Authority Control Vulnerability Audit

Content

The MarketG1.repayBadDebt function is only called by the STRXG1.reservePayBadDebt function, but MarketG1.repayBadDebt does not use the onlyStrx modifier for authentication, and the business design needs to be confirmed with the developer.

Code location: strx-protocol/src/MarketG1.sol

```
function repayBadDebt() external payable {
        require(msg.value > 0, "msg.value is zero");


        uint256 repay = badDebt - badDebtCovered;
        if (repay > msg.value) {
            repay = msg.value;
        }


        if (repay > 0) {
            uint256 covered = badDebtCovered;
            covered += repay;
            badDebtCovered = covered;
            emit BadDebtCovered(repay, covered);
        }


        if (msg.value > repay) {
            payable(msg.sender).transfer(msg.value - repay);
        }
```

```
    }
```

Code location: strx-protocol/src/STRXG1.sol

```
function reservePayBadDebt(uint256 amount) external onlyReserveAdmin {

        require(amount > 0, "pay badDebt amount should be larger than 0");
        require(address(market) != address(0), "market is empty");

        uint256 totalReserves = reserves;
        uint256 claimed = claimedReserves;
        require(totalReserves >= claimed + amount, "reserves not enough");

        uint256 gap = market.badDebtGap();
        require(gap > 0, "badDebt gap is zero");
        if (amount > gap) {
            amount = gap;
        }

        claimed += amount;
        claimedReserves = claimed;

        market.repayBadDebt{value: amount}();
        emit ReservesClaimed(msg.sender, true, amount, claimed);
    }
```

Solution

It is recommended to clarify the call logic of the business and add onlyStrx modifier to MarketG1.repayBadDebt.

Status

Acknowledged; The project team responded that this was designed this way because the MarketG1.repayBadDebt method supports everyone to call to repay bad debts for the Market, not limited to STRX. Calling this method to repay bad debts in the market does not make a profit, but only provides the possibility of repaying bad debts from assets other than strx's reserves.

[N5] [Suggestion] resourceType missing check

Category: Design Logic Audit

Content

The value of resourceType was not checked, the contract code allows a resourceType of 0 or 1.

Code location: strx-protocol/src/MarketG1.sol

```
function totalDelegatedOfType(uint256 resourceType) public view returns (uint256) {
        return address(strx).totalDelegatedResource(resourceType);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
  function delegatableOfType(uint256 resourceType) public view returns (uint256) {
        return address(strx).delegatableResource(resourceType);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
 function totalUnfreezableOfType(uint256 resourceType) public view returns (uint256)
{
        return address(strx).unfreezableBalanceV2(resourceType);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
function maxRentableOfType(uint256 resourceType) public view returns (uint256) {


        uint256 delegatedResource = totalDelegated();

        uint256 undelegatedResource = totalUnfreezable();

        uint256 toUnfreeze = strx.getBalanceToUnfreeze();

        uint256 _budget = (delegatedResource + undelegatedResource - toUnfreeze) *
maxRentalScaleLimit / SCALE;
        if (_budget <= delegatedResource) {
            return 0;
        }
        _budget -= delegatedResource;

        uint256 _delegatable = address(strx).delegatableResource(resourceType);

        return _delegatable < _budget ? _delegatable : _budget;
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
function _liquidateRate(uint256 resourceType) public view returns (uint256) {
        uint256 stableRate = _stableRate(resourceType);
        uint256 currentRate = _rentalRate(0, resourceType);
        return stableRate > currentRate ? currentRate : stableRate;
    }

    function _stableRate(uint256 resourceType) public view returns (uint256) {
        RentRate memory _futureRentRate = futureRentRate[resourceType];
        if (block.timestamp >= uint256(_futureRentRate.timestamp)) {
            return uint256(_futureRentRate.rate);
        }

        RentRate memory _initialRentRate = initialRentRate[resourceType];

        uint256 sums = uint256(_initialRentRate.rate) *
(uint256(_futureRentRate.timestamp) - block.timestamp)
            + uint256(_futureRentRate.rate) * (block.timestamp -
uint256(_initialRentRate.timestamp));
        uint256 time_interval = uint256(_futureRentRate.timestamp -
_initialRentRate.timestamp);

        return sums / time_interval;
    }

    function _rentalRate(uint256 amount, uint256 resourceType) public view returns
(uint256) {
        IRateModel rateModel = rentalRateModels[resourceType];
        if (address(rateModel) == address(0x00)) {
            return 0;
        }

        return rateModel.getRentalRate(
            totalFrozenOfType(resourceType), totalDelegatedOfType(resourceType) +
amount);
    }
```

Solution

It is recommended to limit the value of resourceType to 0 or 1.

Error data will be returned if the resourceType passed in when these functions are called uses a value other than 0 or 1.

Status

Acknowledged; The project team responded that the pre-compiled contracts related to resourceType on chain return 0 when encountering the currently non-existent resourceType. The contracts remain in the chain-like mechanism here, which is reasonable and the contracts neither return any wrong data nor mislead users.

[N6] [Suggestion] Business logic is unclear

Category: Design Logic Audit

Content

The comment of the code indicates that resourceType is 0 or 1, and the logic of onlyBandwidthOrEnergy also allows resourceType to be 0 or 1, but supportsResourceType only returns true after resourceType == 1.

Code location: strx-protocol/src/MarketG1.sol

```
    // resourceType: 0 - bandwidth, 1 - energy
    function supportsResourceType(uint256 resourceType) external view returns (bool)
{
        return resourceType == 1;
    }
```

The onlyBandwidthOrEnergy modifier allows the value of _resourceType to be 0 or 1. But require(_resourceType == 1, "only energy rent market is implemented"); However, only _resourceType is allowed to be 1. The service logic is not clear and rentPaused[0] = true; but there is no code to set rentPaused[0] to false

Code location: strx-protocol/src/MarketG1.sol

```
function setRentPaused(uint256 _resourceType, bool _pause) public onlyAdmin
onlyBandwidthOrEnergy(_resourceType) {
        require(_resourceType == 1, "only energy rent market is implemented");
        rentPaused[_resourceType] = _pause;

        emit RentPaused(_resourceType, _pause);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
function init(
        address _strx,
        IRateModel[2] calldata _rentalRateModels,
        uint256 _maxRentalScaleLimit,
        uint256 _minFee,
        uint256 _feeRatio
    )
        external
        onlyAdmin
    {
        require(!initialized, "already initialized");
        require(_strx != address(0), "strx is empty");
        require(_maxRentalScaleLimit <= SCALE, "setMaxRentalScaleLimit: invalid
rental limit");

        strx = ISTRXForMarket(_strx);
        rentalRateModels = _rentalRateModels;

        initialized = true;
        maxRentalScaleLimit = _maxRentalScaleLimit; // 1e18: 100%
        minFee = _minFee; // 200 * 1e6: 200TRX
        feeRatio = _feeRatio; // 1e16: 1%, fee = MAX(minFee, 1% * rentAmount)

        // pause the rent of bandwidth at init
        rentPaused[0] = true;
```

The initialization is not assigned to liquidateThreshold, and there is no subsequent function to modify the value of liquidateThreshold, so the default value of liquidateThreshold is 0, which will affect the calculation of the following functions.This may lead to calculation errors.

Code location: strx-protocol/src/MarketV1Storage.sol

```
contract MarketV1Storage is AdminStorage {
...
    uint256 public liquidateThreshold;
```

Code location: strx-protocol/src/MarketG1.sol

```
function getRentInfo(
        address renter,
        address receiver,
        uint256 resourceType
    )
        external
        view
        onlyBandwidthOrEnergy(resourceType)
        returns (uint256, uint256, bool)
    {
       ……

        uint256 oneDayRent = rentInfo.amount * rate * 1 days / SCALE;
        uint256 threshold = rentInfo.amount * rate * liquidateThreshold / SCALE;

        uint256 _fee = _calculateFee(rentInfo.amount);
        if (_securityDeposit <= _fee + oneDayRent + threshold) {
            liquidatable = true;
        }

        return (_securityDeposit, _globalRentIndex, liquidatable);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
function rentResource(
        address receiver,
        uint256 amount,
        uint256 resourceType
    )
        external
        payable
        onlyRentNotPaused(resourceType)
        onlyBandwidthOrEnergy(resourceType)
        canRent(receiver, amount, resourceType)
    {
       ……
            uint256 required = rentInfo.amount * rate * (1 days + liquidateThreshold)
/ SCALE + fee;
            require(rentInfo.securityDeposit > required, "resource rent: Not enough
security deposit");
        }
        if (amount > 0) {
            try strx.marketDelegateResource(payable(receiver), amount, resourceType)
{
            } catch {
                revert("resource rent: delegate resource error");
            }
        }
        setRental(msg.sender, receiver, resourceType, rentInfo);
        _updateRentRate(resourceType);

        emit ResourceRented(msg.sender, receiver, resourceType, amount);
        emit RentResource(msg.sender, receiver, amount, resourceType, msg.value,
rentInfo.amount,
            rentInfo.securityDeposit, rentInfo.rentIndex);
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
    function _liquidateCheck(
        RentalInfo memory rentInfo,
        uint256 resourceType
    )
        private
        view
    {
        require(rentInfo.amount > 0, "liquidate: no resource to liquidate");
        uint256 rate = _liquidateRate(resourceType);
        uint256 oneDayRent = rentInfo.amount * rate * 1 days / SCALE;
        uint256 threshold = rentInfo.amount * rate * liquidateThreshold / SCALE;

        uint256 _fee = _calculateFee(rentInfo.amount);
        require(
            rentInfo.securityDeposit <= _fee + oneDayRent + threshold,
            "liquidate: liquidate condition not met"
        );
    }
```

Solution

It is recommended to clarify the code logic of supportResourceType and setRentPaused, and clarify the specific business design to ensure that the liquidateThreshold variable is correctly assigned and used correctly.

Status

Acknowledged; The project team responded that supportsResourceType, which describes the resource type supported by the current contract. Only energy leases are currently open. The project team responded that the rental function for the bandwidth market is not supported at this stage, so the setRentPaused method also supports the suspension/opening of energy types. The project team responded that liquidateThreshold is to calculate the reserved part outside the guaranteed amount of the order. According to the current demand, the reserved part is 0, so the initial value is 0, and there are no extra-cost gas settings.

[N7] [Suggestion] Business logic issue

Category: Design Logic Audit

Content

require(totalReserves >= claimed + amount, "reserves not enough"); should be placed after if (amount > gap) {amount = gap;}.

Code location: strx-protocol/src/STRXG1.sol

```
function reservePayBadDebt(uint256 amount) external onlyReserveAdmin {

        require(amount > 0, "pay badDebt amount should be larger than 0");
        require(address(market) != address(0), "market is empty");

        uint256 totalReserves = reserves;
        uint256 claimed = claimedReserves;
        require(totalReserves >= claimed + amount, "reserves not enough");

        uint256 gap = market.badDebtGap();
        require(gap > 0, "badDebt gap is zero");
        if (amount > gap) {
            amount = gap;
        }

        claimed += amount;
        claimedReserves = claimed;

        market.repayBadDebt{value: amount}();
        emit ReservesClaimed(msg.sender, true, amount, claimed);
    }
```

Solution

It is recommended to clarify the business logic of the code.

Status

Acknowledged; The project team responded that require(totalReserves >= claimed + amount, "reserves not enough");

It is to verify whether the input parameter value of ReserveAdmin is legal.

Taking a smaller value from the following two is to pay off the bad debts in full amount. This sequential logic is relatively clear.

If it is modified, it will cause ReserveAdmin to be confused about the meaning of the incoming parameters because the input parameters cannot be verified.

[N8] [Suggestion] Code optimization

Category: Others

Content

The updateRentRate function is placed under the admin management functions, but the modifier of onlyAdmin is not used.

Code location: strx-protocol/src/MarketG1.sol

```
// -------------------------------- admin management functions ----------------------
----------
    function updateRentRate() public {
        _updateRentRate(0);
        _updateRentRate(1);
    }
```

Solution

The updateRentRate function seems to allow public calls. It is recommended to remove it from admin management functions.

Status

Acknowledged; The project team responded that this issue was only related to comments not code, since the contract has already been deployed, it will not be modified for now.

[N9] [Medium] Excessive authority issue

Category: Authority Control Vulnerability Audit

Content

Admin can set the address of rentalRateModels. If the external contract has not been security audited, or the external contract is set as a malicious contract, the user's funds may be stolen.

Admin can set the value of minFee, feeRatio, but minFee, feeRatio does not limit the maximum and minimum values.

setMaxRentalScaleLimit function has no event record.

Code location: strx-protocol/src/MarketG1.sol

MarketG1.setMaxRentalScaleLimit

MarketG1.setBandwidthRentalRateModel

MarketG1.setEnergyRentalRateModel

MarketG1.setRateModel

MarketG1.setMinFee

MarketG1.setFeeRatio

The admin can set the values of RentFactor, RewardFactor, _reserveAdmin. There is an issue of excessive permissions.

Code location: strx-protocol/src/STRXG1.sol

```
 function setReserveFactors(uint256 _newRentFactor, uint256 _newRewardFactor)
external onlyAdmin _updateIndex {
        require(_newRentFactor <= SCALE, "rent factor exceed scale limit!");
        require(_newRewardFactor <= SCALE, "reward factor exceed scale limit!");

        ReserveFactor memory factor = reserveFactor;

        uint256 oldRentFactor = uint256(factor.rentFactor);
        if (oldRentFactor != _newRentFactor) {
            factor.rentFactor = uint128(_newRentFactor);
            emit RentFactorUpdated(oldRentFactor, _newRentFactor);
        }

        uint256 oldRewardFactor = uint256(factor.rewardFactor);
        if (oldRewardFactor != _newRewardFactor) {
            factor.rewardFactor = uint128(_newRewardFactor);
            emit RewardFactorUpdated(oldRewardFactor, _newRewardFactor);
        }

        reserveFactor = factor;
    }



    function setReserveAdmin(address payable _reserveAdmin) external onlyAdmin {
        address oldReserveAdmin = reserveAdmin;
        reserveAdmin = _reserveAdmin;

        emit ReserveAdminUpdated(oldReserveAdmin, _reserveAdmin);
    }
```

voteOperator can transfer its permissions to other users without the consent of admin.

Code location: strx-protocol/src/STRXG1.sol

```
    function setVoteOperator(address _voteOperator) external onlyVoteOperatorOrAdmin
 {
        emit VoteOperatorUpdated(voteOperator, _voteOperator);
        voteOperator = _voteOperator;
    }
```

ReserveAdmin can call claimReserves and reservePayBadDebt functions at will.

Code location: strx-protocol/src/STRXG1.sol

```
    function claimReserves(uint256 amount) external onlyReserveAdmin {
        require(amount > 0, "claim amount should be larger than 0");
        uint256 totalReserves = reserves;
        uint256 claimed = claimedReserves;
        require(totalReserves >= claimed + amount, "reserves not enough");

        claimed += amount;
        claimedReserves = claimed;

        address payable receiver = reserveAdmin;
        _doTransferOut(receiver, amount);
        emit ReservesClaimed(receiver, false, amount, claimed);
    }
```

Code location: strx-protocol/src/STRXG1.sol

```
   function reservePayBadDebt(uint256 amount) external onlyReserveAdmin {

        require(amount > 0, "pay badDebt amount should be larger than 0");
        require(address(market) != address(0), "market is empty");

        uint256 totalReserves = reserves;
        uint256 claimed = claimedReserves;
        require(totalReserves >= claimed + amount, "reserves not enough");

        uint256 gap = market.badDebtGap();
        require(gap > 0, "badDebt gap is zero");
        if (amount > gap) {
            amount = gap;
        }

        claimed += amount;
        claimedReserves = claimed;

        market.repayBadDebt{value: amount}();
        emit ReservesClaimed(msg.sender, true, amount, claimed);
    }
```

Solution

It is recommended to transfer the authority to the governance contract and use timelock for restrictions, at least using multi-sign contracts for management.

Status

Acknowledged; The project team responded that ReserveAdmin would be transferred to governance and timelock, and that VoteOperator permissions are divided through another contract VoteManager (address: TBx8avtCuPiuYCQfRq97JLpmdnxr2hWR9L). The permission to set voting configurations (including the witness list and the voting ratio) and replace strx VoteOperator would be transferred to governance and timelock for restriction, while the operator ability could only vote following the witness list and voting ratio in the VoteManager contract.

[N10] [Suggestion] init lacks parameter validity checking

Category: Others

Content

The init function does not check the address(0) of _reserveAdmin, _voteOperator.

Code location: strx-protocol/src/STRXG1.sol

```
function init(
        address _market,
        address payable _reserveAdmin,
        address _voteOperator,
        uint256 _rentFactor,
        uint256 _rewardFactor,
        uint256 _maxAmountForEnergy
    )
        external
        payable
        onlyAdmin
    {
        require(!initialized, "only initialize once");
        require(msg.value >= ONE_TRX, "init must provide at least 1 TRX");
        initialized = true;

        round = 1;

        uint256 _delayDays = getUnfreezeDelayDays();
        uint256 unfreezeTimeSpan = _delayDays * 1 days / CHAIN_UNFREEZE_MAX_TIMES;
        unfreezeCalmDownTime = block.timestamp + unfreezeTimeSpan;

        ReserveFactor memory factor = ReserveFactor(uint128(_rentFactor),
uint128(_rewardFactor));
        reserveFactor = factor;
        require(_market != address(0), "market can't be empty");
        market = IMarketForSTRX(_market);
        reserveAdmin = _reserveAdmin;
        voteOperator = _voteOperator;
        minFreezeAmount = ONE_TRX;
        maxAmountForEnergy = _maxAmountForEnergy;

        setClaimable(0);
        setWithdrawal(0);
        setBalanceToFreeze(0);
        setBalanceToUnfreeze(0);

        _deposit(address(this), msg.value);
    }
```

Solution

It is recommended to check whether the parameters are legal during initialization.

Status

Acknowledged

[N11] [Suggestion] Lack of access control

Category: Authority Control Vulnerability Audit

Content

There is no check whether strx or market is a whitelisted address, and no access rights are set. Any user can call the _become function to the input strx or market parameter, which may be malicious.

Code location: strx-protocol/src/STRXG1.sol

```
    function _become(IProxy strx) external {
        require(msg.sender == strx.admin(), "only admin can change brains");
        strx._acceptImplementation();
    }
```

Code location: strx-protocol/src/MarketG1.sol

```
    function _become(IProxy market) external {
        require(msg.sender == market.admin(), "only admin can change brains");
        market._acceptImplementation();
    }
```

Solution

It is recommended to add access restrictions such as: only the admin of real strx and market can call the _become function.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002305050001	SlowMist Security Team	2023.04.24 - 2023.05.05	Medium 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 1 medium risk, and 10 suggestion vulnerabilities. The code has been deployed to the mainnet.

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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist