Security Audit Report for Synclub-Contracts

Date: May 8, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About Target Contracts 1

1.2 Disclaimer 1

1.3 Procedure of Auditing 2

1.3.1 Software Security 2

1.3.2 DeFi Security 2

1.3.3 NFT Security 3

1.3.4 Additional Recommendation 3

1.4 Security Model 3

Chapter 2 Findings **5**

2.1 DeFi Security 5

2.1.1 Potential DoS due to unrestricted withdrawal amount 5

2.1.2 Logic error in function claimUndelegated() 7

2.1.3 Incorrect check in function compoundRewards() 9

2.1.4 Timely compoundRewards() when calculating shares 10

2.1.5 Incorrect reward due to logic error 12

2.2 Note 14

2.2.1 Potential centralization risk 14

Report Manifest

Item	Description
Client	Lista
Target	Synclub-Contracts

Version History

Version	Date	Description
1.0	May 8, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-05-09 10:22:43+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Synclub-Contracts[1] of Lista. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `Synclub-Contracts` folder contract only. Specifically, the files covered in this audit include:

```
ListaStakeManager.sol
```

Listing 1.1: Audit Scope for this Report

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Synclub-Contracts	Version 1	87189aa8358df3ae84b266ad4231e4aaf80df368
	Version 2	c51a8fc10355933daa98692dbb671e618a4b63d0

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

[1] https://github.com/lista-dao/synclub-contracts

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*High*	High	Medium
	Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **five** potential issues and **one** note as follows:

- Medium Risk: 3
- Low Risk: 2
- Note: 1

ID	Severity	Description	Category	Status
1	Medium	Potential DoS due to unrestricted withdrawal amount	Defi Security	Fixed
2	Medium	Logic error in function claimUndelegated()	Defi Security	Fixed
3	Low	Incorrect check in function compoundRewards()	Defi Security	Fixed
4	Low	Timely compoundRewards() when calculating shares	Defi Security	Confirmed
5	Medium	Incorrect reward due to logic error	Defi Security	Fixed
6	-	Potential centralization risk	Note	

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Potential DoS due to unrestricted withdrawal amount

Severity Medium

Status Fixed in Version 2

Introduced by Version 1

Description The function undelegateFrom() in the ListaStakeManager contract undelegates assets from the STAKE_HUB based on user withdrawal requests. The function getAmountToUndelegate() iterates through the withdrawalQueue and calculates the total amount to be withdrawn. However, the function requestWithdraw() does not impose a minimum limit on the withdrawal amount. Malicious users could frequently invoke requestWithdraw() with minimal cost, thereby increasing the length of the withdrawalQueue and potentially launch the DoS attack.

```
function requestWithdraw(uint256 _amountInSlisBnb)
    external
    override
    whenNotPaused
{
    require(_amountInSlisBnb > 0, "Invalid Amount");


     uint256 bnbToWithdraw = convertSnBnbToBnb(_amountInSlisBnb);
    require(bnbToWithdraw > 0, "Bnb amount is too small");


```

```
         requestUUID++;
        userWithdrawalRequests[msg.sender].push(
            WithdrawalRequest({
                uuid: requestUUID,
                amountInSnBnb: _amountInSlisBnb,
                startTime: block.timestamp
            })
        );


         withdrawalQueue.push(
            UserRequest({
                uuid: requestUUID,
                amount: bnbToWithdraw,
                amountInSlisBnb: _amountInSlisBnb
            })
        );
        requestIndexMap[requestUUID] = withdrawalQueue.length - 1;


         IERC20Upgradeable(slisBnb).safeTransferFrom(
            msg.sender,
            address(this),
            _amountInSlisBnb
        );
        emit RequestWithdraw(msg.sender, _amountInSlisBnb);
    }
```

Listing 2.1: ListaStakeManager.sol

```
    function undelegateFrom(address _operator, uint256 _amount)
        external
        override
        whenNotPaused
        onlyRole(BOT)
        returns (uint256)
    {
        require(totalSnBnbToBurn == 0, "Old requests should be processed first");
        require(_amount <= (getAmountToUndelegate() + reserveAmount), "Given bnb amount is too large");
        uint256 _shares = convertBnbToShares(_operator, _amount);
        uint256 _actualBnbAmount = convertSharesToBnb(_operator, _shares);


        unbondingBnb += _actualBnbAmount;
        IStakeHub(STAKE_HUB).undelegate(bscValidator, _shares);


        emit UndelegateFrom(_operator, _actualBnbAmount, _shares);
        return getAmountToUndelegate();
    }
```

Listing 2.2: ListaStakeManager.sol

```
function getAmountToUndelegate() public view override returns (uint256 _amountToUndelegate) {
    if (nextUndelegatedRequestIndex == withdrawalQueue.length) {
        return 0;
    }
    uint256 totalAmountToWithdraw = 0;
    for (uint256 i = nextUndelegatedRequestIndex; i < withdrawalQueue.length; ++i) {
        UserRequest storage req = withdrawalQueue[i];
        uint256 amount = req.amount;
        totalAmountToWithdraw += amount;
    }


    _amountToUndelegate = totalAmountToWithdraw - unbondingBnb;
}
```

Listing 2.3: ListaStakeManager.sol

Impact The function `undelegateFrom()` will not be able to execute properly.

Suggestion Add minimum value checks for both deposit and withdrawal operations.

2.1.2 Logic error in function claimUndelegated()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `claimUndelegated()` is used to claim undelegated assets from `STAKE_HUB`. Lines 370-381 are designed to distribute assets for withdrawal requests made before the contract upgrade. Since the amount in the `claim()` function cannot be predetermined, it may trigger the logic within Lines 372-375. In this scenario, the function will return directly, ceasing further execution of the code below. This results in the corresponding shares not being burned and `totalDelegated` not being updated, which is incorrect.

```
345 function claimUndelegated(address _validator)
346     external
347     override
348     whenNotPaused
349     onlyRole(BOT)
350     returns (uint256 _uuid, uint256 _amount)
351 {
352     require(totalSnBnbToBurn == 0, "Old request not undelegated yet");
353
354
355     uint256 balanceBefore = address(this).balance;
356     IStakeHub(STAKE_HUB).claim(_validator, 0);
357     require(address(this).balance > balanceBefore, "Nothing to claim");
358     uint256 undelegatedAmount = address(this).balance - balanceBefore;
359
360
361     undelegatedQuota += undelegatedAmount;
```

```
        unbondingBnb -= undelegatedAmount;


        uint256 coveredAmount = 0;
        uint256 coveredSlisBnbAmount = 0;
        uint256 oldLastUUID = requestUUID;


        if (withdrawalQueue.length != 0) {
            oldLastUUID = withdrawalQueue[0].uuid - 1;
        }


        for (uint256 i = nextConfirmedRequestUUID; i <= oldLastUUID; ++i) {
            BotUndelegateRequest storage botRequest = uuidToBotUndelegateRequestMap[i];
            if (undelegatedQuota < botRequest.amount) {
                emit ClaimUndelegatedFrom(_validator, nextConfirmedRequestUUID, undelegatedAmount);
                return (nextConfirmedRequestUUID, undelegatedAmount);
            }
            botRequest.endTime = block.timestamp;
            undelegatedQuota -= botRequest.amount;
            coveredAmount += botRequest.amount;
            coveredSlisBnbAmount += botRequest.amountInSnBnb;
            ++nextConfirmedRequestUUID;
        }


        // new logic
        for (uint256 i = nextConfirmedRequestUUID; i <= requestUUID; ++i) {
            UserRequest storage req = withdrawalQueue[requestIndexMap[i]];
            if (req.uuid == 0 || req.amount > undelegatedQuota) {
                break;
            }
            undelegatedQuota -= req.amount;
            coveredAmount += req.amount;
            coveredSlisBnbAmount += req.amountInSlisBnb;
            ++nextConfirmedRequestUUID;
        }


        totalDelegated -= coveredAmount;
        if (coveredSlisBnbAmount > 0) {
            ISLisBNB(slisBnb).burn(address(this), coveredSlisBnbAmount);
        }


        _uuid = nextConfirmedRequestUUID;
        _amount = undelegatedAmount;


        emit ClaimUndelegatedFrom(_validator, _uuid, _amount);
    }
```

Listing 2.4: ListaStakeManager.sol

Impact The share price in the protocol is miscalculated.

Suggestion Revise the logic to ensure key global variables are correctly updated.

2.1.3 Incorrect check in function compoundRewards()

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `compoundRewards()` first retrieves the amount of `BNB` held by the validator, then subtracts the contract's recorded `totalDelegated` to compute the result as a reward, a portion of which is allocated as a fee. Each execution of `compoundRewards()` accumulates previous fees as `totalFee`, which is then converted into shares through the function `claimFee()` and minted to the `revenuePool`. Since the reward has not been undelegated, `totalFee` is not withdrawn from `STAKE_HUB`, thus `totalFee` also generates rewards. The check at Line 853 does not account for `totalFee`, which is incorrect.

```
846   function compoundRewards()
847       external
848       override
849       whenNotPaused
850       onlyRole(BOT)
851   {
852       require(totalDelegated > 0, "No funds delegated");
853
854
855       uint256 totalBNBInValidators = getTotalBnbInValidators();
856       require(totalBNBInValidators >= totalDelegated && totalBNBInValidators - totalDelegated >
              totalFee, "No new fee to compound");
857       uint256 totalProfit = totalBNBInValidators - totalDelegated - totalFee;
858       uint256 fee = 0;
859       if (synFee > 0) {
860           fee = totalProfit * synFee / TEN_DECIMALS;
861           totalFee += fee;
862       }
863       uint256 totalUserProfit = totalProfit - fee;
864
865
866       totalDelegated += totalUserProfit;
867
868
869       emit RewardsCompounded(fee);
870   }
```

Listing 2.5: ListaStakeManager.sol

Impact Variable `totalFee` also generates rewards, which results in the loss of these rewards.

Suggestion Revise the logic to ensure that rewards generated by `totalFee` can also be claimed.

2.1.4 Timely compoundRewards() when calculating shares

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The protocol provides `BNB` as incentive rewards for staking the `BNB`. The rewards are distributed to staking users in proportion to their share of `LP` (i.e., `slisBnb`) tokens in the pool. However, in function `deposit()`, before calculating the shares to mint, rewards are not timely distributed. In this case, the rewards that originally belonged to previous stakers have been allocated to new stakers, which is unfair. The similar issue also exists in functions `requestWithdraw()` and `setSynFee()`.

```
function deposit() external payable override whenNotPaused {
    uint256 amount = msg.value;
    require(amount > 0, "Invalid Amount");


    uint256 slisBnbToMint = convertBnbToSnBnb(amount);
    require(slisBnbToMint > 0, "Invalid SlisBnb Amount");
    amountToDelegate += amount;


    ISLisBNB(slisBnb).mint(msg.sender, slisBnbToMint);


    emit Deposit(msg.sender, msg.value);
}
```

Listing 2.6: ListaStakeManager.sol

```
function requestWithdraw(uint256 _amountInSlisBnb)
    external
    override
    whenNotPaused
{
    require(_amountInSlisBnb > 0, "Invalid Amount");


    uint256 bnbToWithdraw = convertSnBnbToBnb(_amountInSlisBnb);
    require(bnbToWithdraw > 0, "Bnb amount is too small");


    requestUUID++;
    userWithdrawalRequests[msg.sender].push(
        WithdrawalRequest({
            uuid: requestUUID,
            amountInSnBnb: _amountInSlisBnb,
            startTime: block.timestamp
        })
    );


    withdrawalQueue.push(
        UserRequest({
            uuid: requestUUID,
            amount: bnbToWithdraw,
            amountInSlisBnb: _amountInSlisBnb
        })
    );
    requestIndexMap[requestUUID] = withdrawalQueue.length - 1;


    IERC20Upgradeable(slisBnb).safeTransferFrom(
        msg.sender,
        address(this),
        _amountInSlisBnb
    );
    emit RequestWithdraw(msg.sender, _amountInSlisBnb);
}
```

Listing 2.7: ListaStakeManager.sol

```
function compoundRewards()
    external
    override
    whenNotPaused
    onlyRole(BOT)
{
    require(totalDelegated > 0, "No funds delegated");

    uint256 totalBNBInValidators = getTotalBnbInValidators();
    require(totalBNBInValidators >= totalDelegated && totalBNBInValidators - totalDelegated >
        totalFee, "No new fee to compound");
    uint256 totalProfit = totalBNBInValidators - totalDelegated - totalFee;
    uint256 fee = 0;
    if (synFee > 0) {
        fee = totalProfit * synFee / TEN_DECIMALS;
        totalFee += fee;
    }
    uint256 totalUserProfit = totalProfit - fee;

    totalDelegated += totalUserProfit;
    emit RewardsCompounded(fee);
}
```

Listing 2.8: ListaStakeManager.sol

```
function setSynFee(uint256 _synFee)
    external
    override
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    require(_synFee <= TEN_DECIMALS, "_synFee must not exceed 10000 (100%)");
    synFee = _synFee;
```

```
        emit SetSynFee(_synFee);
    }
```

Listing 2.9: ListaStakeManager.sol

Impact Rewards are not distributed timely.

Suggestion Invoke the function `compoundRewards()` before calculating `shares`/`bnbToWithdraw` or setting new `synFee`.

Feedback from the project It's better to follow the original design considering the gas cost introduced by frequent user operations. We learned that `STAKE_HUB` distributes rewards on a daily basis, so we have cron job executing `compoundRewards()` everyday.

2.1.5 Incorrect reward due to logic error

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `compoundRewards()` calculates rewards based on the difference between the assets delegated to the validator and `totalDelegated`. In the function `claimUndelegated()`, the actual distributed `coveredAmount` is used to calculate the corresponding `coveredSlisBnbAmount` as shares, which are then burned, and `totalDelegated` is decreased by the `coveredAmount`. However, the `coveredAmount` may be less than the assets actually claimed for undelegated. Therefore, `totalDelegated` may exceed its actual value, leading to incorrect reward calculations at Line 856 in function `compoundRewards()`.

```
846   function compoundRewards()
847       external
848       override
849       whenNotPaused
850       onlyRole(BOT)
851   {
852       require(totalDelegated > 0, "No funds delegated");
853
854
855       uint256 totalBNBInValidators = getTotalBnbInValidators();
856       require(totalBNBInValidators >= totalDelegated && totalBNBInValidators - totalDelegated >
              totalFee, "No new fee to compound");
857       uint256 totalProfit = totalBNBInValidators - totalDelegated - totalFee;
858       uint256 fee = 0;
859       if (synFee > 0) {
860           fee = totalProfit * synFee / TEN_DECIMALS;
861           totalFee += fee;
862       }
863       uint256 totalUserProfit = totalProfit - fee;
864
865
866       totalDelegated += totalUserProfit;
867
868
```

```
    emit RewardsCompounded(fee);
}
```

Listing 2.10: ListaStakeManager.sol

```
function claimUndelegated(address _validator)
    external
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _uuid, uint256 _amount)
{
    require(totalSnBnbToBurn == 0, "Old request not undelegated yet");


    uint256 balanceBefore = address(this).balance;
    IStakeHub(STAKE_HUB).claim(_validator, 0);
    require(address(this).balance > balanceBefore, "Nothing to claim");
    uint256 undelegatedAmount = address(this).balance - balanceBefore;


    undelegatedQuota += undelegatedAmount;
    unbondingBnb -= undelegatedAmount;


    uint256 coveredAmount = 0;
    uint256 coveredSlisBnbAmount = 0;
    uint256 oldLastUUID = requestUUID;


    if (withdrawalQueue.length != 0) {
        oldLastUUID = withdrawalQueue[0].uuid - 1;
    }


    for (uint256 i = nextConfirmedRequestUUID; i <= oldLastUUID; ++i) {
        BotUndelegateRequest storage botRequest = uuidToBotUndelegateRequestMap[i];
        if (undelegatedQuota < botRequest.amount) {
            emit ClaimUndelegatedFrom(_validator, nextConfirmedRequestUUID, undelegatedAmount);
            return (nextConfirmedRequestUUID, undelegatedAmount);
        }
        botRequest.endTime = block.timestamp;
        undelegatedQuota -= botRequest.amount;
        coveredAmount += botRequest.amount;
        coveredSlisBnbAmount += botRequest.amountInSnBnb;
        ++nextConfirmedRequestUUID;
    }


    // new logic
    for (uint256 i = nextConfirmedRequestUUID; i <= requestUUID; ++i) {
        UserRequest storage req = withdrawalQueue[requestIndexMap[i]];
        if (req.uuid == 0 || req.amount > undelegatedQuota) {
                break;
            }
            undelegatedQuota -= req.amount;
            coveredAmount += req.amount;
            coveredSlisBnbAmount += req.amountInSlisBnb;
            ++nextConfirmedRequestUUID;
        }


        totalDelegated -= coveredAmount;
        if (coveredSlisBnbAmount > 0) {
            ISLisBNB(slisBnb).burn(address(this), coveredSlisBnbAmount);
        }


        _uuid = nextConfirmedRequestUUID;
        _amount = undelegatedAmount;


        emit ClaimUndelegatedFrom(_validator, _uuid, _amount);
    }
```

Listing 2.11: ListaStakeManager.sol

Impact The parameter `totalDelegated` does not match the actual value, resulting in an incorrect reward calculation.

Suggestion Revise the logic to ensure that `totalDelegated` is accurately updated.

2.2 Note

2.2.1 Potential centralization risk

Introduced by `Version 1`

Description There are some centralization risks in this protocol. For example, `whitelistValidator()` is a privileged function and can only be accessed by `DEFAULT_ADMIN_ROLE`. More precisely, the contract's `administrator` can set any address as a `validator` with function `whitelistValidator()`. Losing the `administrator`'s corresponding private key can potentially result in the loss of user assets. It is suggested to consider the multi-signature or securely managing the private key.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS