NSR

Audit Report

Thu May 29 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

NSR Audit Report

1 Executive Summary

1.1 Project Information

Description	A lending protocol based on NSR and SSR
Type	Lending
Auditors	ScaleBit
Timeline	Thu May 08 2025 - Thu May 29 2025
Languages	Solidity
Platform	BSC
Methods	Architecture Review, Unit Testing, Manual Review

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
NSR	contracts/NSR.sol	caa3bb8736a76b58a6b09445feeda29b88ac3295
SSR	contracts/SSR.sol	492dc47463db23221a63bc1420df90a231251fbf
CON	contracts/config.sol	5d90381f8ffa92762e8c6e498d34be5337ec6f4f
NSR1	contracts/NSRdefi.sol	3f14ad6528874268851e012abf86096bae19b73c

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	9	5	4
Informational	1	0	1
Minor	4	1	3
Medium	2	2	0
Major	2	2	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by NSR to identify any potential issues and vulnerabilities in the source code of the NSR smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 9 issues of varying severity, listed below.

ID	Title	Severity	Status
NSR-1	Missing Validation for `Stake_ratio` Parameter	Major	Fixed
NSR-2	Missing Status Check for `reply_swap2NSR()`	Medium	Fixed
NSR-3	Use `safeTransfer()` and `safeTransferFrom()` Instead of `transfer()` and `transferFrom()`	Medium	Fixed
NSR-4	Missing Check for `percent`	Minor	Fixed
NSR-5	The `withdraw_debet_u()` Process can be Optimized	Minor	Acknowledged
NSR-6	Missing a Check to Ensure that the Amount to be Deducted is less than or equal to the Available Balance before Deduction	Minor	Acknowledged
NSR-7	Remove Unnecessary Steps	Informational	Acknowledged

SSR-1	`SSRToken.transfer` Will Always Fail	Major	Fixed
NSR1-1	SafeMath Is Redundant	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the NSR Smart Contract :

Owner

- `setowner` : Set the admin address
- `setUfee` : Set withdrawal fee ratio
- `set_depositfee` : Set deposit fee ratio
- `set_protrol_ratio` : Set protocol fee ratio
- `add_rent` : Add new lending asset
- `add_stake` : Add new staking asset
- `set_rent` : Update lending asset parameters
- `set_stake` : Update staking asset parameters
- `set_ratio_debet_SSR2NSR` : Set SSR to NSR conversion ratio
- `set_withdraw_ratio` : Set withdrawal ratio
- `reply_swap2NSR` : Approves/rejects staking application
- `reply_NSR2lend` : Approves/rejects lending asset conversion application
- `withdraw_NSR_owner` : Withdraw NSR fees

User

- `Swap_lend2NSR` : Convert lending asset to NSR
- `apply_stake2NSR` : Apply to stake assets
- `make_debet_order` : Create borrowing order
- `order_accept` : Accept borrowing order
- `apply_2_lend` : Convert NSR to lending asset
- `expose` : Liquidate overdue loans

- `withdraw_debet_u`: Withdraw borrowed assets
- `withdraw_u`: Withdraw NSR token
- `redeem`: Repay loan and retrieve staked assets

4 Findings

NSR-1 Missing Validation for `Stake_ratio` Parameter

Severity: Major

Status: Fixed

Code Location:

contracts/NSRdefi.sol#387-394

Descriptions:

In the `make_debet_order` function, the borrowable asset amount is calculated based on the user-provided `Stake_ratio`:

```
function make_debet_order(uint percent, uint time, uint Stake_ratio) external returns
(bool, uint) {
    require(time == 7 || time == 14 || time == 30 || time == 60);  // Fixed durations
    require(Stake_ratio <= 100);  // Only upper bound checked

    debet memory order;
    person_account storage personal = balances[msg.sender];

    order.debeter = msg.sender;
    order.amount = personal.debet_NSR * Stake_ratio / ACCURACY1;
    ...
}
```

Currently, the only check performed is `Stake_ratio <= 100`, which allows values like 99 or 98. This is problematic, as a borrower could create an order with an unreasonably low stake ratio, and if they fail to repay, the lender would suffer losses during liquidation.

```
require(IWETH(info[i].source).transfer(order.lender, order.stake_amtList[i] * 90 / 100),
"1");
require(IWETH(info[i].source).transfer(admin, order.stake_amtList[i] -
order.stake_amtList[i] * 90 / 100), "1");
totalsupply_weth[info[i].source] -= order.stake_amtList[i];
```

Such behavior could lead to undercollateralized loans and unfair outcomes for lenders.

Suggestion:

`Stake_ratio` should be strictly validated against a protocol-defined maximum value (e.g., 70%,) to prevent users from setting dangerously low ratios.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NSR-2 Missing Status Check for `reply_swap2NSR()`

Severity: Medium

Status: Fixed

Code Location:

contracts/NSRdefi.sol#349-370

Descriptions:

The `reply_swap2NSR()` function is used to approve or reject a user's `stake2NSR` request based on the `replyid`.

```
function reply_swap2NSR(uint indexid, uint replyid) public onlyowner returns(bool){
    swapindex memory info = swap_check[indexid];
    stake_info memory s_info = stake_list[info.index];  //
    person_account storage personal = balances[info.applyer];
    if (replyid==0){
        require(IWETH(s_info.source).transfer(info.applyer,info.amount),"1");  //     eth
        info.state = checkState.REJECT;
    }else if (replyid==1) {
        require(SSRToken.mint(info.SSRamount));  //mint
        require(SSRToken.transfer(info.applyer,info.SSRamount),"1");  //
        personal.stake2NSR_amount[s_info.source] += info.amount;
        personal.debet_NSR += info.NSRdebet;  //       NSR
        personal.SSRamt += info.SSRamount;
        require(SSRToken.transferFrom(info.applyer,address(this),info.SSRamount),"1");
        require(SSRToken.burn(info.SSRamount),"3");
        info.state = checkState.PASS;
    }
    info.reply_time = block.timestamp;
    swap_check[indexid] = info;  //update
    emit
Stake2NSR(indexid,info.applyer,replyid,info.index,info.amount,info.NSRdebet,info.reply_tim
    return true;
}
```

However, the protocol does not verify the current status of the `swap_check` entry. As a result, even if the status is already `REJECT` or `PASS`, the process can still be executed again.

Suggestion:

It is recommended to add a status check.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NSR-3 Use `safeTransfer()` and `safeTransferFrom()` Instead of `transfer()` and `transferFrom()`

Severity: Medium

Status: Fixed

Code Location:

contracts/NSRdefi.sol#304

Descriptions:

Some non-standard tokens do not return a bool on BEP20 methods. This will make the call break, making it impossible to use these tokens.

```
function Swap_lend2NSR(uint index, uint256 amount) external returns(uint,uint){
//u2NSR
    rent_info memory info = rent_list[index]; //

    require(IBEP20(info.source).transferFrom(msg.sender, address(this), amount),"1");  //

    total_supply[info.source] += amount; //total supply
    // balances[msg.sender].lend_amount[info.source] -= amount;  //
    uint amount_out = transfer_dynamic(amount,rent_list[index].dynamic, NSRdynamic);
//       NSR
    uint NSRAmount = amount_out * info.ratio_2_deposit_NSR / ACCURACY2;
    uint fee = NSRAmount * depositfee / ACCURACY2;
    balances[address(this)].deposit_NSR += fee;
    require(NSRToken.mint(NSRAmount));  //mint
    require(NSRToken.transfer(msg.sender,NSRAmount - fee),"1");  //
    emit lend2NSR(msg.sender,index,amount,NSRAmount - fee,
fee,info.ratio_2_deposit_NSR, block.timestamp);
    return (amount,NSRAmount);    //     u                    NSR
}
```

Suggestion:

Use `SafeTransferLib` or `SafeERC20`, replace transfer with `safeTransfer()` and `transferFrom()` with `safeTransferFrom()` when transferring `BEP20` tokens.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NSR-4 Missing Check for `percent`

Severity: Minor

Status: Fixed

Code Location:

contracts/NSRdefi.sol#389

Descriptions:

In the `make_debet_order()` function, the protocol starts by verifying that `Stake_ratio <= 100`.

```
function make_debet_order(uint percent,uint time, uint Stake_ratio) external
returns(bool, uint) {   //only use weth after transfered
    require(time==7 || time == 14 || time==30 || time == 60);  //
    require(Stake_ratio <= 100);
    debet memory order;   //
    person_account storage personal = balances[msg.sender];
```

However, it does not validate whether the value is a proper percentage within the expected range.

Suggestion:

It is recommended to add a full validation check to ensure the `percent` represents a valid value.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NSR-5 The `withdraw_debet_u()` Process can be Optimized

Severity: Minor

Status: Acknowledged

Code Location:

contracts/NSRdefi.sol#488-490

Descriptions:

In the `withdraw_debet_u()` function, the protocol transfers `order.temp_NSR - withdraw_fee` NSR tokens to `msg.sender`, then immediately transfers the same amount back from the user to the protocol and burns it.

```
   uint U_amount = (order.temp_NSR - withdraw_fee) * ACCURACY2 /
rent_list[index].ratio_2_deposit_NSR;  //u amount
   U_amount = transfer_dynamic(U_amount,NSRdynamic,rent_list[index].dynamic);  //
  U
   require(NSRToken.transfer(msg.sender,order.temp_NSR - withdraw_fee),"1");  //
sender
   require(NSRToken.transferFrom(msg.sender,address(this),order.temp_NSR -
withdraw_fee),"1");   //sender
   require(NSRToken.burn(order.temp_NSR - withdraw_fee),"3");  //
```

This round-trip transfer is unnecessary. The protocol can directly burn the tokens without transferring them to and from the user, which would simplify the logic and save gas.

Suggestion:

It is recommended to optimize this process.

NSR-6 Missing a Check to Ensure that the Amount to be Deducted is less than or equal to the Available Balance before Deduction

Severity: Minor

Status: Acknowledged

Code Location:

contracts/NSRdefi.sol#569-581

Descriptions:

In the `withdraw_u()` function, the protocol directly transfers tokens to the recipient and then subtracts the amount from `deposit_NSR_interest` or `deposit_NSR`.

```
if(typet ==1){//interest
      require(balance.deposit_NSR_interest > 0);
      require(NSRToken.transfer(accept,amount),"1");  //
      balance.deposit_NSR_interest -= amount;
      amount_out = amount;

   }else if(typet==2){ //deposit
      require(balance.deposit_NSR > 0);
      require(NSRToken.transfer(accept,amount),"1");  //
      balance.deposit_NSR -= amount;
      amount_out = amount;
   }
```

However, it is better to first check whether `deposit_NSR_interest` or `deposit_NSR` is greater than or equal to the withdrawal amount to avoid underflow or inconsistent state.

Suggestion:

It is recommended to check whether `deposit_NSR_interest` or `deposit_NSR` is greater than or equal to the withdrawal amount to avoid underflow or inconsistent state.

NSR-7 Remove Unnecessary Steps

Severity: Informational

Status: Acknowledged

Code Location:

contracts/NSRdefi.sol#357-364

Descriptions:

In the `reply_swap2NSR()` function, when `replyid == 1`, the protocol mints SSR tokens and transfers them to the applyer, then immediately transfers the same amount of tokens back from the applyer and burns them. This process appears to be redundant and ineffective, as it results in no net token gain or utility.

```
require(SSRToken.mint(info.SSRamount));  //mint
        require(SSRToken.transfer(info.applyer,info.SSRamount),"1");  //
        personal.stake2NSR_amount[s_info.source] += info.amount;
        personal.debet_NSR += info.NSRdebet;  //       NSR
        personal.SSRamt += info.SSRamount;
        require(SSRToken.transferFrom(info.applyer,address(this),info.SSRamount),"1");
        require(SSRToken.burn(info.SSRamount),"3");
        info.state = checkState.PASS;
```

Suggestion:

It is recommended to remove unnecessary steps to optimize the process.

SSR-1 `SSRToken.transfer` Will Always Fail

Severity: Major

Status: Fixed

Code Location:

contracts/SSR.sol#83-88

Descriptions:

The `transfer` function in the `SSRToken` contract includes the following logic:

```
function transfer(address recipient, uint256 amount) external returns (bool) {
    NSRdefi(defi_addr).transferSSR(_msgSender(), recipient, amount);
    _transfer(_msgSender(), recipient, amount);
    return true;
}
```

However, the `NSRdefi` contract does **not** implement a `transferSSR` function. As a result, every call to `transfer` will revert due to the missing external call, causing the entire function to fail.

Suggestion:

Remove the call to `transferSSR` or ensure that the `NSRdefi` contract correctly implements this function to avoid unnecessary reverts and broken token transfers.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NSR1-1 SafeMath Is Redundant

Severity: Minor

Status: Acknowledged

Code Location:

contracts/NSR.sol#344

Descriptions:

Starting from Solidity version 0.8, overflow is automatically checked, and the code provided uses Solidity version ^0.8.0, so using SafeMath is unnecessary and will consume extra gas.

```
  using SafeMath for uint256;
```

```
pragma solidity 0.8.19;
```

Suggestion:

It is recommended to directly use addition, subtraction, multiplication, and division operations without using SafeMath.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer