BEOSIN
Blockchain Security

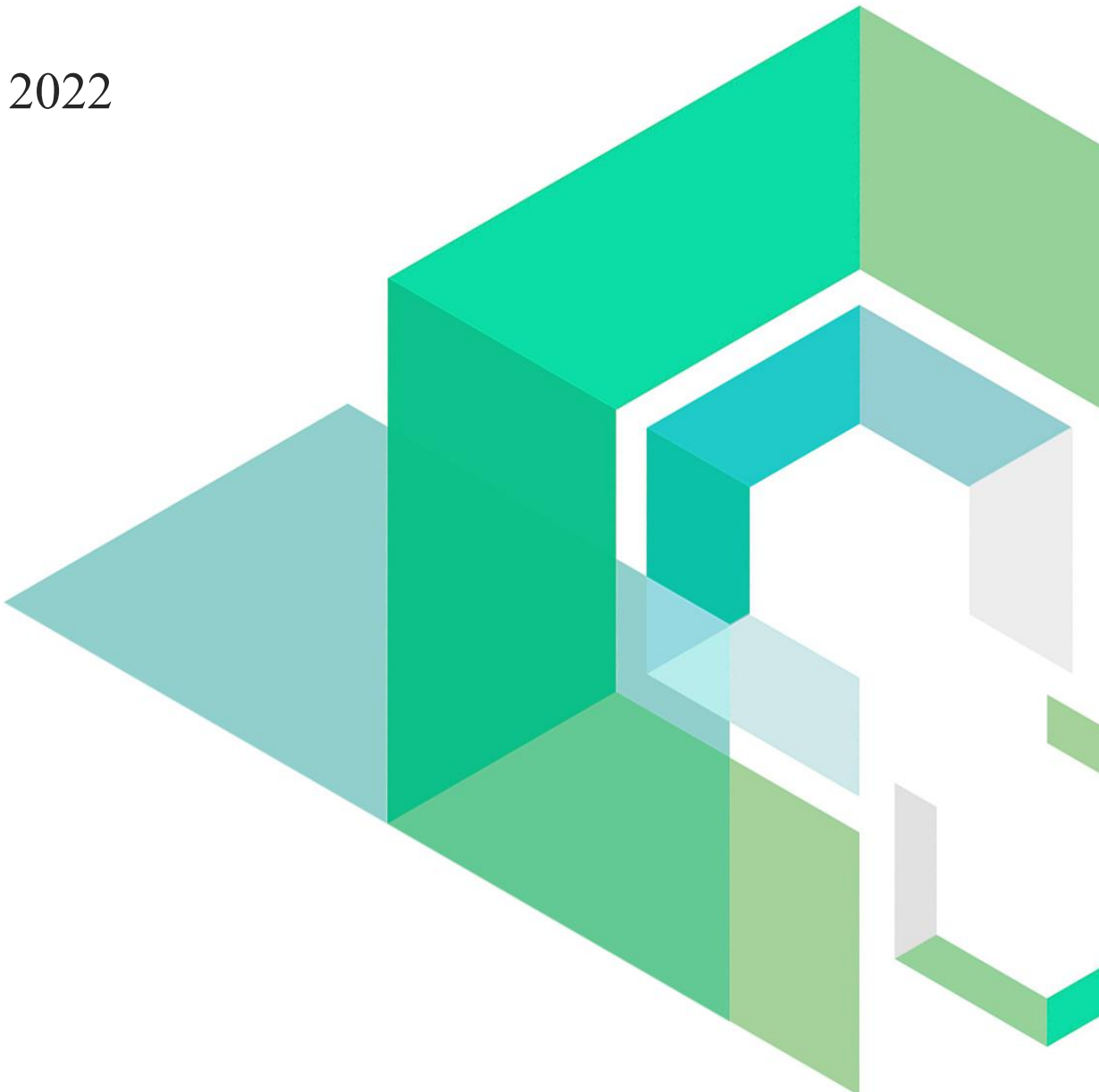
BNB

Smart Contract Security Audit

V1.0

No. 202204141830

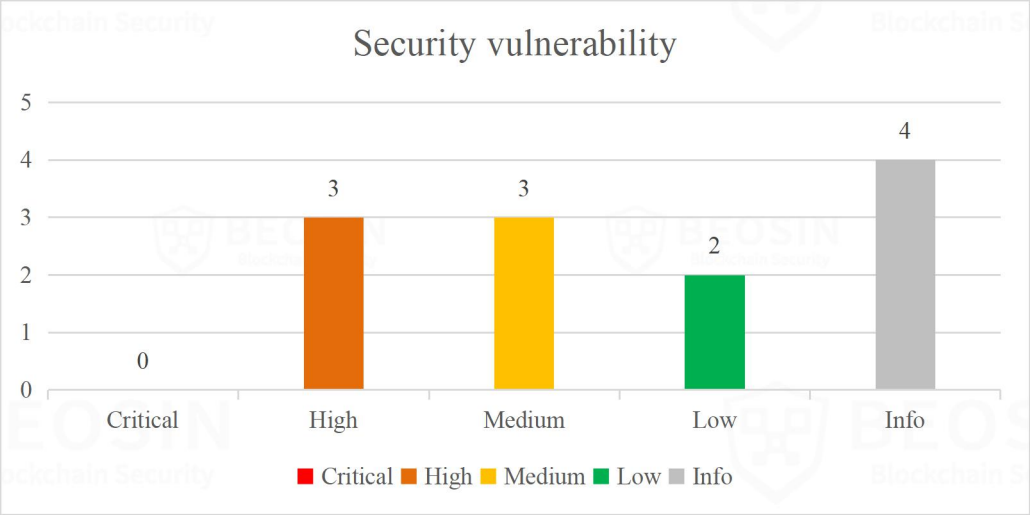
Apr 14th, 2022



Contents

Summary of audit results

After auditing, 3 High-risks, 3 Medium-risks, 2 Low-risks and 4 Info items were identified in the BNB project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Risk Description:**

1. Operator has a high authority

In the BinancePool_R3 contract, when a user withdraws his tokens, he needs to call the unstake function to initiate the request, and the operator address will call the *distributeRewards* function before he can withdraw them successfully. If the operator address does not call the *distributeRewards* function properly, the user may not be able to withdraw his tokens.

2. Centralization risk

The owner in the BNB project can set the operator address, and both the owner and operator addresses can modify key parameters in the contract. For some parameters, only the owner or operator has the permission to modify. There may be some centralization risk.

3. Event trigger parameter error

The event parameter triggered in the *mintBonds* function in the aBNBb_R2 contract is the number of bonds, while in the mint function it is shares. There will be no security risk, but it will cause confusion to users, and the number of transfers queried will be inconsistent with the actual number of transfers.

- **Project Description:**

1. Basic Token Information

Token name	Ankr BNB Reward Earning Bond
Token symbol	aBNBb
Decimals	18
Total supply	Initial supply is 0 (Mintale, burnable)
Token type	BEP-20

2. Business overview

The BNB project contains a token contract and a business contract. In the token contract, the number of shares is recorded in the contract, and the user queries the number of bonds. Shares and bonds are converted according to a certain ratio (the ratio can be arbitrarily modified by the owner or operator address). Users can exchange aBNBb tokens and aBNBc tokens in the aBNBb token contract (a fee may be charged). Users can stake BNB tokens in the BinancePool contract to get aBNBb tokens. The BNB staked by the user will be sent to the tokenHub contract. The aBNBc token contract and tokenHub contract are not within the scope of this audit. When the user withdraws the BNB tokens staked by the BinancePool contract, it needs to be operated by the operator address to successfully withdraw. The amount extracted by the user must exceed the set minimum amount (can be changed by the operator).

1 Overview

1.1 Project Overview

Project Name	BNB	
Platform	BNB Chain	
File Hash (SHA256)	ABNBb_R1.sol	1c2176b3f0738f59d5937476b9490cfbbbea530561c93ebdcf153987b7edf414 (Initial)
	ABNBb_R2.sol	635677d27f5b4d8ef7062c9463b42edd6169e94e8dceb637b3ef6ecd67419983 (Final)
	BinancePool_R2.sol	0e6c2f57befaf4e55aa428df413c9a295b3ab93d90375baed53202da6e465cc4 (Initial)
	BinancePool_R4.sol	d4c5a4dc582cc8c5df2726dd2f3d5aae31fab3595482e8740163bd9d5489df6a (Final)

1.2 Audit Overview

Audit work duration: March 10th, 2022 – April 14th, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Technology Co. Ltd.

2 Findings

Index	Risk description	Severity level	Status
BNB-1	Operator has a high authority	High	Acknowledged
BNB-2	Owner has a high authority	High	Fixed
BNB-3	_claimersForManualDistribute update exception	High	Fixed
BNB-4	stashedForManualDistributes update exception	Medium	Fixed
BNB-5	Can't claim funds manually	Medium	Fixed
BNB-6	Bypass operator for token withdrawal	Medium	Fixed
BNB-7	Centralization risk	Low	Acknowledged
BNB-8	Some staking tokens may cannot be withdrawn	Low	Acknowledged
BNB-9	Event not triggered	Info	Fixed
BNB-10	Event trigger parameter error	Info	Partially Fixed
BNB-11	The event is triggered repeatedly	Info	Fixed
BNB-12	Redundant code	Info	Fixed

Risk Details Description:

1. BNB-1 is not fixed and may cause users to be unable to withdraw assets.

2. BNB-7 is not fixed and may cause a potential centralization risk.

3. BNB-8 is not fixed and may cause users to be unable to withdraw some small assets.

4. BNB-10 is not fully fixed but does not cause security issues.

[BNB-1] Operator has a high authority

Severity Level	**High**
Type	Business Security
Lines	BinancePool_R2.sol#L57-60, L120-190
Description	In the BinancePool contract, after the user initiates a withdrawal request, an Operator operation is required for the withdrawal to succeed. If there is a problem with the operator address, the user may not be able to withdraw properly.

```
modifier onlyOperator() {
    require(msg.sender == _operator, "sender is not an operator");
    _;
}
```

Figure 1 Source code of *onlyOperator* modifier

```
function distributeRewards() external payable nonReentrant onlyOperator {
    require(
        address(this).balance >= _minimumStake,
        "must be greater than min unstake amount"
    );
    address[] memory claimers = new address[](
        _pendingClaimers.length - _pendingGap
    );
    uint256[] memory amounts = new uint256[](
        _pendingClaimers.length - _pendingGap
    );
    uint256 poolBalance = address(this).balance;
    uint256 j = 0;
    uint256 gaps = 0;
    uint256 i = 0;
```

Figure 2 Partly source code of *distributeRewards* function

Recommendations	It is recommended to change it to allow users to withdraw their funds directly.
Status	Acknowledged. The project team replies anyway an operator has to send BNB to the pool. **However, if the operator does not call the *distributeRewards* function correctly, the user may not be able to withdraw his tokens.**

[BNB-2] Owner has a high authority

Severity Level	**High**
Type	Business Security
Lines	ABNBb_R1.sol#L65-79, L148-164, L186-198
Description	In the aBNBb_R1 contract, the owner can call *mintBonds* and *mint* functions to mint tokens at will, and call *burn* and *burnAndSetPending* functions to burn tokens at any address.

```
modifier onlyMinter() {
    require(
        msg.sender == owner() || msg.sender == _crossChainBridge,
        "Minter: not allowed"
    );
    _;
}

modifier onlyBondMinter() {
    require(
        msg.sender == owner() || msg.sender == _binancePool,
        "Minter: not allowed"
    );
    _;
}
```

Figure 3 Source code of *onlyMinter* and *onlyBondMinter* modifiers (Unfixed)

```
function mintBonds(address account, uint256 amount)
    public
    override
    onlyBondMinter
{
    _totalStaked += amount;
    uint256 shares = _bondsToShares(amount);
    _mint(account, shares);
    emit Transfer(address(0), account, amount);
}

function mint(address account, uint256 shares) public onlyMinter {
    _lockedShares = _lockedShares - int256(shares);
    _mint(account, shares);
    emit Transfer(address(0), account, shares);
}

function burn(address account, uint256 amount) public override onlyMinter {
    uint256 shares = _bondsToShares(amount);
    _lockedShares = _lockedShares + int256(shares);
    _burn(account, shares);
    emit Transfer(account, address(0), amount);
}
```

Figure 4 Source code of related functions

```
function burnAndSetPending(address account, uint256 amount)
    public
    override
    onlyBondMinter
{
    _pendingBurn[account] = _pendingBurn[account] + amount;
    _pendingBurnsTotal = _pendingBurnsTotal + amount;
    uint256 sharesToBurn = _bondsToShares(amount);
    _totalUnbondedBonds += amount;
    _burn(account, sharesToBurn);
    emit Transfer(address(0), account, amount);
}
```

Figure 5 Source code of *burnAndSetPending* function

Recommendations	It is recommended to cancel the corresponding permissions of owner in the modifiers.
Status	Fixed.

```
modifier onlyMinter() {
    require(msg.sender == _crossChainBridge, "Minter: not allowed");
    _;
}

modifier onlyBondMinter() {
    require(msg.sender == _binancePool, "Minter: not allowed");
    _;
}
```

Figure 6 Source code of *onlyMinter* and *onlyBondMinter* modifiers (Fixed)

[BNB-3] _claimersForManualDistribute update exception

Severity Level	**High**
Type	Business Security
Lines	BinancePool_R4.sol#L261-290
Description	When there is already a record of sending failure, the sending operation is still performed again. And after the sending fails, the data in pendingClaimerUnstakes is deleted, and then the user calls the *distributeManual* function to receive 0. In addition, the user data in _claimersForManualDistribute is not modified in the *distributeManual* function. Users may not be able to call the *unstake*, *unstakeCerts* and *unstakeBonds* functions again.

```
function distributeManual(uint256 id) external override nonReentrant {
    require(
        markedForManualDistribute[id],
        "not marked for manual distributing"
    );
    address[] memory claimers = new address[](1);
    uint256[] memory amounts = new uint256[](1);

    address claimer = _pendingClaimers[id];
    address payable wallet = payable(claimer);
    uint256 amount = pendingClaimerUnstakes[claimer];

    markedForManualDistribute[id] = false;
    stashedForManualDistributes -= amount;

    require(
        address(this).balance >= amount + stashedForManualDistributes,
        "insufficient pool balance"
    );
    claimers[0] = claimer;
    amounts[0] = amount;
    IBondToken(_bondContract).updatePendingBurning(claimer, amount);
    pendingClaimerUnstakes[claimer] = 0;

    (bool result, ) = wallet.call{value: amount}("");
    require(result, "failed to send rewards to claimer");
    delete _pendingClaimers[id];

    emit RewardsDistributed(claimers, amounts, 0);
}
```

Figure 7 Partly source code of *distributeManual* function (Unfixed)

Recommendations	It is recommended to first judge whether there is a record of sending failure, and if so, proceed directly to the next cycle. In the *distributeManual* function, change the

	user data in _claimersForManualDistribute to false.
Status	Fixed.

```
function distributeManual(uint256 id) external override nonReentrant {
    require(
        markedForManualDistribute[id],
        "not marked for manual distributing"
    );
    address[] memory claimers = new address[](1);
    uint256[] memory amounts = new uint256[](1);

    address claimer = _pendingClaimers[id];
    address payable wallet = payable(claimer);
    uint256 amount = pendingClaimerUnstakes[claimer];

    markedForManualDistribute[id] = false;
    _claimersForManualDistribute[claimer] = false;
    stashedForManualDistributes -= amount;

    require(
        address(this).balance >= amount + stashedForManualDistributes,
        "insufficient pool balance"
    );
    claimers[0] = claimer;
    amounts[0] = amount;
    IBondToken(_bondContract).updatePendingBurning(claimer, amount);
    pendingClaimerUnstakes[claimer] = 0;

    (bool result, ) = wallet.call{value: amount}("");
    require(result, "failed to send rewards to claimer");
    delete _pendingClaimers[id];

    emit RewardsDistributed(claimers, amounts, 0);
}
```

Figure 8 Partly source code of *distributeManual* function (Fixed)

[BNB-4] stashedForManualDistributes update exception

Severity Level	Medium
Type	Business Security
Lines	BinancePool_R2.sol#L120-190
Description	When traversing in the *distributeRewards* function, it is not determined whether the user is in a manual claim state. The data in _pendingClaimers and pendingClaimerUnstakes of users who are in the manual claim state are not deleted. If the operator calls the *calcPendingGap* or *resetPendingGap* function and then calls the *distributeRewards* function to traverse _pendingClaimers again, the users who were in the manual claiming state will be processed again. It may cause the data of users who were in the state of manual collection before to be added to the variable stashedForManualDistributes again.

```
for (i = _pendingGap; i < _pendingClaimers.length; i++) {
    if (poolBalance == 0) {
        break;
    }
    address claimer = _pendingClaimers[i];
    uint256 toDistribute = pendingClaimerUnstakes[claimer];
    /* we might have gaps lets just skip them (we shrink them on full claim) */
    if (claimer == address(0) || toDistribute == 0) {
        gaps++;
        continue;
    }
    if (poolBalance < toDistribute) {
        toDistribute = poolBalance;
    }
    address payable wallet = payable(address(claimer));
    (bool result, ) = wallet.call{value: toDistribute, gas: 40000}("");
    /* when we delete items from array we generate new gap, lets remember how many gaps we did to skip them in next claim */
    if (!result) {
        gaps++;
        markedForManualDistribute[i] = true;
        stashedForManualDistributes += toDistribute;
        emit ManualDistributeExpected(claimer, toDistribute, i);
        continue;
    }
```

Figure 9 Partly source code of *distributeRewards* function (Unfixed)

Recommendations	It is recommended to judge whether it is in a manual claiming state when sending tokens to users.
Status	Fixed.

```
for (i = _pendingGap; i < _pendingClaimers.length; i++) {
    if (poolBalance == 0 || j > maxClaimers) {
        break;
    }
    address claimer = _pendingClaimers[i];
    if (_claimersForManualDistribute[claimer]) {
        continue;
    }
    uint256 toDistribute = pendingClaimerUnstakes[claimer];
    /* we might have gaps lets just skip them (we shrink them on full claim) */
    if (claimer == address(0) || toDistribute == 0) {
        gaps++;
        continue;
    }
    if (poolBalance < toDistribute) {
        toDistribute = poolBalance;
    }
    address payable wallet = payable(address(claimer));
    (bool result, ) = wallet.call{value: toDistribute, gas: 10000}("");
    /* when we delete items from array we generate new gap, lets remember how many gaps we did to skip them in next claim */
    if (!result) {
        gaps++;
        markedForManualDistribute[i] = true;
        _claimersForManualDistribute[claimer] = true;
        stashedForManualDistributes += toDistribute;
        emit ManualDistributeExpected(claimer, toDistribute, i);
        continue;
    }
```

Figure 10 Partly source code of *distributeRewards* function (Fixed)

[BNB-5] Can't claim funds manually

Severity Level	Medium
Type	Business Security
Lines	BinancePool_R2.sol#L120-221
Description	Only the *distributeRewards* function can receive the rewards transferred by the operator, and each time *distributeRewards* may consume all the BNB in the contract. It may cause that there is not enough BNB in the contract for the *distributeManual* function to claim.

```
function distributeRewards() external payable nonReentrant onlyOperator {
    require(
        address(this).balance >= _minimumStake,
        "must be greater than min unstake amount"
    );
    address[] memory claimers = new address[](
        _pendingClaimers.length - _pendingGap
    );
    uint256[] memory amounts = new uint256[](
        _pendingClaimers.length - _pendingGap
    );
    uint256 poolBalance = address(this).balance;
    uint256 j = 0;
    uint256 gaps = 0;
    uint256 i = 0;
```

Figure 11 Partly source code of *distributeRewards* function (Unfixed)

Recommendations	It is recommended to subtract the value of the variable stashedForManualDistributes when getting poolBalance.
Status	Fixed.

```
function distributeRewards(uint256 maxClaimers)
    external
    payable
    override
    nonReentrant
    onlyOperator
{
    uint256 poolBalance = address(this).balance -
        stashedForManualDistributes;
    require(
        poolBalance >= _minimumStake,
        "must be greater than min unstake amount"
    );
    address[] memory claimers = new address[](
        _pendingClaimers.length - _pendingGap
    );
    uint256[] memory amounts = new uint256[](
        _pendingClaimers.length - _pendingGap
    );
    uint256 j = 0;
    uint256 gaps = 0;
    uint256 i = 0;
```

Figure 12 Partly source code of *distributeRewards* function (Fixed)

[BNB-6] Bypass operator for token withdrawal

Severity Level	Medium
Type	Business Security
Lines	BinancePool_R2.sol#L101-118, L192-221
Description	In the *unstakeCerts*, *unstakeBonds* and *unstake* functions of the BinancePool_R3 contract, it is not determined whether the user is in the state of manual claiming. If the user is in the state of manual claiming, he can submit a withdrawal request through the *unstakeCerts*, *unstakeBonds* and *unstake* functions, bypassing the operator to claim directly. The quotas of other users will be consumed by users who claim them abnormally, so that other users who need to claim them manually cannot claim them normally.

```
function unstake(uint256 amount) external nonReentrant {
    require(
        amount >= _minimumStake,
        "value must be greater than min unstake amount"
    );
    require(
        IERC20Upgradeable(_bondContract).balanceOf(msg.sender) >= amount,
        "cannot unstake more than have on address"
    );
    if (pendingClaimerUnstakes[msg.sender] == 0) {
        _pendingClaimers.push(msg.sender);
    }
    pendingClaimerUnstakes[msg.sender] =
        pendingClaimerUnstakes[msg.sender] +
        amount;
    IInternetBond(_bondContract).burnAndSetPending(msg.sender, amount);
    emit UnstakePending(msg.sender, amount);
}
```

Figure 13 Source code of *unstake* function (Unfixed)

```
function distributeManual(uint256 id) external nonReentrant {
    require(
        markedForManualDistribute[id],
        "not marked for manual distributing"
    );
    address[] memory claimers = new address[](1);
    uint256[] memory amounts = new uint256[](1);

    address claimer = _pendingClaimers[id];
    address payable wallet = payable(address(claimer));
    uint256 amount = pendingClaimerUnstakes[claimer];

    markedForManualDistribute[id] = false;
    stashedForManualDistributes -= amount;

    require(
        address(this).balance >= amount + stashedForManualDistributes,
        "insufficient pool balance"
    );
    claimers[0] = claimer;
    amounts[0] = amount;
    IInternetBond(_bondContract).updatePendingBurning(claimer, amount);
    pendingClaimerUnstakes[claimer] = 0;

    (bool result, ) = wallet.call{value: amount}("");
    require(result, "failed to send rewards to claimer");
    delete _pendingClaimers[id];

    emit RewardsDistributed(claimers, amounts, 0);
}
```

Figure 14 Source code of *distributeManual* function (Unfixed)

Recommendations	It is recommended to modify the *unstakeCerts*, *unstakeBonds* and *unstake* functions when users can manually claim tokens.
Status	Fixed.

```
function unstake(uint256 amount) external override badClaimer nonReentrant {
    require(
        amount >= _minimumStake,
        "value must be greater than min unstake amount"
    );
    require(
        ICertToken(_bondContract).balanceOf(msg.sender) >= amount,
        "cannot unstake more than have on address"
    );
    if (pendingClaimerUnstakes[msg.sender] == 0) {
        _pendingClaimers.push(msg.sender);
    }
    pendingClaimerUnstakes[msg.sender] += amount;
    IBondToken(_bondContract).burnAndSetPending(msg.sender, amount);
    emit UnstakePending(msg.sender, amount);
    emit isRebasing(true);
}
```

Figure 15 Source code of *unstake* function (Fixed)

```
function distributeManual(uint256 id) external override nonReentrant {
    require(
        markedForManualDistribute[id],
        "not marked for manual distributing"
    );
    address[] memory claimers = new address[](1);
    uint256[] memory amounts = new uint256[](1);

    address claimer = _pendingClaimers[id];
    address payable wallet = payable(claimer);
    uint256 amount = pendingClaimerUnstakes[claimer];

    markedForManualDistribute[id] = false;
    _claimersForManualDistribute[claimer] = false;
    stashedForManualDistributes -= amount;

    require(
        address(this).balance >= amount + stashedForManualDistributes,
        "insufficient pool balance"
    );
    claimers[0] = claimer;
    amounts[0] = amount;
    IBondToken(_bondContract).updatePendingBurning(claimer, amount);
    pendingClaimerUnstakes[claimer] = 0;

    (bool result, ) = wallet.call{value: amount}("");
    require(result, "failed to send rewards to claimer");
    delete _pendingClaimers[id];

    emit RewardsDistributed(claimers, amounts, 0);
}
```

Figure 16 Source code of *distributeManual* function (Fixed)

[BNB-7] Centralization risk

Severity Level	**Low**
Type	Business Security
Lines	ABNBb_R1.sol#L100-103, L260-270 BinancePool_R2.sol#L235-245, L251-253, L263-273
Description	In the aBNBb_R1 contract, the owner can call functions such as *changeOperator* and *changeBinancePool* to modify the related parameters of the contract, which may cause a certain risk of centralization.

```
function changeOperator(address operator) public onlyOwner {
    _operator = operator;
}

function changeBinancePool(address binancePool) public onlyOwner {
    _binancePool = binancePool;
}

function changeCrossChainBridge(address crossChainBridge) public onlyOwner {
    _crossChainBridge = crossChainBridge;
}
```

Figure 17 Source code of related functions

In the BinancePool contract, the owner can call the *resetPendingGap*, *setMinimumStake*, *changeIntermediary*, *changeBondContract* and *changeTokenHub* functions to modify the contract-related parameters, which may lead to certain centralization risks.

```
function changeIntermediary(address intermediary) external onlyOwner {
    _intermediary = intermediary;
}

function changeBondContract(address bondContract) external onlyOwner {
    _bondContract = bondContract;
}

function changeTokenHub(address tokenHub) external onlyOwner {
    _tokenHub = ITokenHub(tokenHub);
}
```

Figure 18 Source code of related functions

Recommendations	It is recommended to use multi-signature wallet, TimeLock contract, DAO, governance contract, etc. as the contract owner.

Status	Acknowledged.

[BNB-8] Some collateral tokens may cannot be withdrawn

Severity Level	Low
Type	Business Security
Lines	BinancePool_R3.sol#L99-151
Description	A lower limit is set when withdrawing in the BinancePool_R3 contract. If the user's remaining stake is less than _minimumStake, withdrawal cannot be performed.

```
function unstake(uint256 amount) external override nonReentrant {
    require(
        amount >= _minimumStake,
        "value must be greater than min unstake amount"
    );
    require(
        ICertToken(_bondContract).balanceOf(msg.sender) >= amount,
        "cannot unstake more than have on address"
    );
    if (pendingClaimerUnstakes[msg.sender] == 0) {
        _pendingClaimers.push(msg.sender);
    }
    pendingClaimerUnstakes[msg.sender] += amount;
    IBondToken(_bondContract).burnAndSetPending(msg.sender, amount);
    emit UnstakePending(msg.sender, amount);
}
```

Figure 19 Source code of unstake function

Recommendations	It is recommended to remove the minimum number of unstake.
Status	Acknowledged. The project team replies Binance has requirement for unstake: min 1 BNB. For example within a week 100 users have ustaked 0.00001 BNB each and bnb backend can not unstake 0.001BNB from validator.

[BNB-9] Event not triggered

Severity Level	Info
Type	Business Security
Lines	ABNBb_R1.sol#L260-270 BinancePool_R2.sol# L251-253, L263-273
Description	The functions *resetPendingGap*, *setMinimumStake*, *changeIntermediary*, *changeBondContract* and *changeTokenHub* in the BinancePool contract did not trigger corresponding events.

```
function changeIntermediary(address intermediary) external onlyOwner {
    _intermediary = intermediary;
}

function changeBondContract(address bondContract) external onlyOwner {
    _bondContract = bondContract;
}

function changeTokenHub(address tokenHub) external onlyOwner {
    _tokenHub = ITokenHub(tokenHub);
}
```

Figure 20 Source code of related functions (Unfixed)

The *changeOperator*, *changeBinancePool* and *changeCrossChainBridge* functions in the aBNBb_R1 contract did not trigger the corresponding events.

```
function changeOperator(address operator) public onlyOwner {
    _operator = operator;
}

function changeBinancePool(address binancePool) public onlyOwner {
    _binancePool = binancePool;
}

function changeCrossChainBridge(address crossChainBridge) public onlyOwner {
    _crossChainBridge = crossChainBridge;
}
```

Figure 21 Source code of related functions (Unfixed)

Recommendations	It is recommended to trigger the corresponding events.
Status	Fixed.

```
function changeBondContract(address bondContract) external onlyOwner {
    _bondContract = bondContract;
    emit BondContractChanged(bondContract);
}

function changeCertContract(address certToken) external onlyOwner {
    _certContract = certToken;
    emit CertContractChanged(certToken);
}

function changeTokenHub(address tokenHub) external onlyOwner {
    _tokenHub = ITokenHub(tokenHub);
    emit IntermediaryChanged(tokenHub);
}
```

Figure 22 Source code of related functions (Fixed)

```
function changeOperator(address operator) external onlyOwner {
    _operator = operator;
    emit OperatorChanged(operator);
}

function changeBinancePool(address binancePool) external onlyOwner {
    _binancePool = binancePool;
    emit BinancePoolChanged(binancePool);
}

function changeCrossChainBridge(address crossChainBridge)
    external
    onlyOwner
{
    _crossChainBridge = crossChainBridge;
    emit CrossChainBridgeChanged(crossChainBridge);
}
```

Figure 23 Source code of related functions (Fixed)

[BNB-10] Event trigger parameter error

Severity Level	Info
Type	Business Security
Lines	ABNBb_R1.sol#130-133, L152-163
Description	The third parameter of the event trigger in the *mint* function in the aBNBb_R1 contract is wrong and not consistent with other places. The first and second parameters of the event triggered in the *burnAndSetPending* function are wrong.

```
function mint(address account, uint256 shares) public onlyMinter {
    _lockedShares = _lockedShares - int256(shares);
    _mint(account, shares);
    emit Transfer(address(0), account, shares);
}
```

Figure 24 Source code of mint function

```
function burnAndSetPending(address account, uint256 amount)
    public
    override
    onlyBondMinter
{
    _pendingBurn[account] = _pendingBurn[account] + amount;
    _pendingBurnsTotal = _pendingBurnsTotal + amount;
    uint256 sharesToBurn = _bondsToShares(amount);
    _totalUnbondedBonds += amount;
    _burn(account, sharesToBurn);
    emit Transfer(address(0), account, amount);
}
```

Figure 25Source code of *burnAndSetPending* function (Unfixed)

Recommendations	It is recommended to convert the third parameter in the event triggered in the *mint* function into bonds, and the first parameter and the second parameter in the *burnAndSetPending* function are interchanged.
Status	Partially Fixed. The third parameter of the event triggered in the *mint* function is not fixed.

```
function burnAndSetPending(address account, uint256 amount)
    external
    override
    onlyBondMinter
{
    _pendingBurn[account] = _pendingBurn[account] + amount;
    _pendingBurnsTotal = _pendingBurnsTotal + amount;
    _totalUnbondedBonds += amount;
    uint256 sharesToBurn = bondsToShares(amount);
    _burn(account, sharesToBurn);
    _aBNBc.burn(address(this), sharesToBurn);
    emit Transfer(account, address(0), amount);
}
```

Figure 26 Source code of *burnAndSetPending* function

[BNB-11] The event is triggered repeatedly

Severity Level	Info
Type	Business Security
Lines	ABNBb_R1.sol#L160-164 ERC20.sol#L251-261
Description	Both mint and _mint functions in the aBNBb_R1 contract trigger the Transfer event, which may cause abnormal display data.

```
function mint(address account, uint256 shares) external onlyMinter {
    _lockedShares = _lockedShares - int256(shares);
    _mint(account, shares);
    emit Transfer(address(0), account, shares);
}
```

Figure 27 Source code of *mint* function

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

Figure 28 Source code of *_mint* function (Unfixed)

Recommendations	It is recommended to delete redundant Transfer events.
Status	Fixed.

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
}
```

Figure 29 Source code of *_mint* function (Fixed)

[BNB-12] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	ABNBb_R1.sol#L27-29, L45
Description	The aBNBb_R1 contract introduces the SafeMathUpgradeable, MathUpgradeable and SignedSafeMathUpgradeable libraries, but the functions in them are not used, and the variable _collectableFee is not used after being defined.

```
contract aBNBb_R1 is OwnableUpgradeable, ERC20Upgradeable, IInternetBond {
    using SafeMathUpgradeable for uint256;
    using MathUpgradeable for uint256;
    using SignedSafeMathUpgradeable for int256;

    /**
     * Variables
     */

    address private _operator;
    address private _crossChainBridge;
    address private _binancePool;
    uint256 private _ratio;
    uint256 private _totalStaked;
    uint256 private _totalUnbondedBonds;
    int256 private _lockedShares;

    mapping(address => uint256) private _pendingBurn;
    uint256 private _pendingBurnsTotal;
    uint256 private _collectableFee;
```

Figure 30 Source code of aBNBb_R1 contract

Recommendations	It is recommended to delete the relevant code.
Status	Fixed.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

Affiliated to BEOSIN Technology Pte. Ltd., BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com