Bitlayer: Security Audit Report

DogScan Security Team

July 16th, 2025

Contents

DogScan Security Audit Report

Project	Bitlayer
Chain	BitLayer Testnet (ID: 200810)
Smart Contract	YBTCStakingVault
Audit Date	July 16th, 2025
Report Version	1.0

1. Executive Summary

The YBTCStakingVault contract has completed security audit. The audit has identified one medium-risk technical issue and some design characteristics that require attention. The technical issue primarily involves accounting inconsistency in the `withdraw` function, which may lead to mismatches between the contract's fund records and actual balance. The design characteristics mainly involve the contract's centralized management mechanisms and price setting processes, including the contract administrator's emergency rescue capabilities and operator-approved withdrawal processes, as well as the operator role's ability to set LP token prices.

The overall risk level is assessed as MEDIUM. It is recommended to fix the accounting inconsistency issue before deployment, and users should carefully understand the contract mechanisms before participating.

2. Audit Scope

The audit scope covers the `YBTCStakingVault` contract at address `0xb2418c6af198e9664c988c8800f334b2cb9` on BitLayer (Chain ID: 200810).

The contract includes the following key components:

- **Main Contract**: `YBTCStakingVault.sol`
- **Inherited Libraries**: OpenZeppelin contracts including ERC20, Pausable, AccessControlEnumerable
- **Dependencies**: SafeERC20, EnumerableSet utilities
- **Interface**: `IYBTC.sol` for native token wrapping/unwrapping

3. Audit Methodology

This audit employed a multi-agent AI security analysis framework. The process involved multiple specialized AI agents performing a comprehensive review of the smart contract's source code and on-chain state. The methodology included:

1. **Automated Vulnerability Detection**: Static analysis agents scanned the code for known vulnerability patterns, logical flaws, and deviations from best practices.
2. **Economic Model Analysis**: DeFi-specialized agents assessed the contract's economic incentives, potential for manipulation, and mathematical soundness.
3. **Access Control Review**: Agents analyzed the role-based access control (RBAC) implementation, identifying privileged functions and potential centralization risks.
4. **Manual Heuristic Review**: A lead security strategist AI synthesized the findings from all agents, triaging alerts, eliminating false positives by cross-referencing the call chain and contract logic, and assessing the systemic impact of vulnerabilities in combination.

4. Findings Summary

ID	Title	Severity	Status
M-01	Accounting Inconsistency Risk	**Medium**	**Recommend Fix**
I-01	Centralized Control Over Funds and Permissioned Withdrawals	**Info**	**User Notice**
I-02	Privileged Role Price Setting Mechanism	**Info**	**User Notice**

5. Detailed Findings

[M-01] Accounting Inconsistency Risk (Medium)

Severity: Medium

Description The contract's `withdraw` function has an accounting inconsistency issue. This function allows addresses with the `MANAGER_ROLE` to withdraw `ybtcToken` from the contract, but does not correspondingly update the `totalStaked` state variable, leading to mismatches between the contract's fund records and actual balance.

Impact This accounting inconsistency may lead to the following issues:

1. **Inaccurate Fund Records**: The `totalStaked` records a staked amount greater than the contract's actual `ybtcToken` balance
2. **Subsequent User Withdrawal Failures**: When other users attempt to withdraw, they may fail due to insufficient contract balance
3. **Economic Model Imbalance**: LP token value calculations may be based on inaccurate `totalStaked` data
4. **Audit Trail Difficulties**: Inconsistency between contract state and actual fund flows makes accurate financial auditing difficult

Code Snippet

```
function withdraw(uint256 amount) public onlyRole(MANAGER_ROLE) {
    require(amount > 0, "Invalid");
    ybtcToken.safeTransfer(msg.sender, amount);  // Transfer funds
    emit Withdraw(msg.sender, amount);
    // Missing: totalStaked -= amount;  // Does not update total staked record
}
```

Compare with the correct implementation in the `exit` function:

```
function exit(uint256 amount) public whenNotPaused onlyRole(MANAGER_ROLE) {
    require(balanceOf(msg.sender) >= amount, "Insufficient LP");
    uint256 ybtcAmount = convertLpToYBTC(amount);
    _burn(msg.sender, amount);
    totalStaked -= ybtcAmount;  // Correctly updates totalStaked
    ybtcToken.safeTransfer(msg.sender, ybtcAmount);
    emit Exit(msg.sender, amount, ybtcAmount);
}
```

Recommendations

1. **Fix the withdraw function**: Update the `totalStaked` state variable while transferring funds:

```
function withdraw(uint256 amount) public onlyRole(MANAGER_ROLE) {
    require(amount > 0, "Invalid");
    require(amount <= ybtcToken.balanceOf(address(this)), "Insufficient balance")
        ;
    require(amount <= totalStaked, "Exceeds total staked");

    totalStaked -= amount;  // Add this line
    ybtcToken.safeTransfer(msg.sender, amount);
    emit Withdraw(msg.sender, amount);
}
```

2. **Add balance checks**: Ensure the withdrawal amount does not exceed the contract's actual balance and recorded total staked amount
3. **Implement accounting audit mechanisms**: Regularly check the consistency between `totalStaked` and actual contract balance

User Notice Users should understand: 1. Administrators may withdraw funds through the `withdraw` function, affecting the contract's fund sufficiency 2. It is recommended to monitor the project's fund management transparency and operational records 3. Evaluate the contract's fund management mechanisms and risk control measures before participating

[I-01] Centralized Control Over Funds and Permissioned Withdrawals (Info)

Severity: Info

Description The contract adopts a centralized management model, including the following design features:

1. **Rescue Function (`rescue`)**: The `rescue` function is controlled by the `DEFAULT_ADMIN_ROLE`, allowing administrators to transfer the contract's `ybtcToken` balance in emergency situations. This is an emergency rescue mechanism.
2. **Manager Withdraw Function (`withdraw`)**: The `withdraw` function allows addresses with the `MANAGER_ROLE` to withdraw `ybtcToken` from the contract for operational and management purposes.

3. **Permissioned Claim Process**: The contract adopts an approval-based withdrawal mechanism. Users need to first call `requestClaim` to create a withdrawal request, then have an address with the `OPERATOR_ROLE` call `approveClaim` for approval, and finally users can complete the withdrawal through `claim` or `claimNative`.

Impact Under this design pattern, the contract adopts a centralized operational mechanism. Users need to understand that:

- Withdrawals require operator approval and are not immediate
- Administrators have emergency rescue and operational management permissions
- This design may be suitable for scenarios requiring compliance review or risk control

Code Snippet

```
function rescue(address to) public onlyRole(DEFAULT_ADMIN_ROLE) {
    require(to != address(0), "Invalid");
    uint256 balance = ybtcToken.balanceOf(address(this));
    ybtcToken.safeTransfer(to, balance);
    emit Rescue(to, balance);
}

function approveClaim(bytes32[] calldata claimIds) public onlyRole(OPERATOR_ROLE)
     {
    for (uint i = 0; i < claimIds.length; i++) {
        require(
            claimRequests[claimIds[i]].user != address(0),
            "Invalid claim ID"
        );
        claimRequests[claimIds[i]].approved = true;
    }
}
```

User Notice Users should understand before participating in this contract:

1. This is a centrally managed staking contract where withdrawals require administrator approval
2. Administrators have emergency fund rescue permissions
3. Users should evaluate the team's credibility and operational transparency
4. It is recommended to pay attention to the project's governance mechanisms and multi-signature management

[I-02] Privileged Role Price Setting Mechanism (Info)

Severity: Info

Description The contract adopts a privileged role-controlled price setting mechanism, including the following design features:

1. **Operator Price Setting**: The `OPERATOR_ROLE` can set the conversion price between LP tokens and the underlying `ybtcToken` through the `setPrice` function. This mechanism includes time interval restrictions and change magnitude limitations.
2. **Manager Exit Mechanism**: The `MANAGER_ROLE` can exit at the current set price through the `exit` function, which is a management function.
3. **Price Constraint Mechanism**: The contract sets a `require(newPrice >= oldPrice)` constraint, ensuring prices can only increase and not decrease.

Impact Under this design pattern:

- Prices are manually set by operators rather than market-driven
- Price setting includes frequency and magnitude restrictions
- The price constraint mechanism may be suitable for specific business scenarios, such as stablecoin staking or special economic models

Code Snippet

```
function setPrice(uint256 newPrice) public onlyRole(OPERATOR_ROLE) {
    require(
        block.timestamp >= lastPriceChangeTime + minPriceChangeDuration,
        "Too soon"
    );
    uint256 currentEpoch = getCurrentEpoch();
    if (nextEpoch > 0) {
        uint256 oldPrice = priceHistory[currentEpoch].price;
        uint256 maxChange = (oldPrice * (100 + maxPriceChangePercentage)) /
            100;
        require(newPrice >= oldPrice, "Price decrease"); // Price can only
          increase
        require(newPrice <= maxChange, "Exceeds max change");
    }
    priceHistory[nextEpoch] = PriceInfo({
        price: newPrice,
        timestamp: block.timestamp
    });
    lastPriceChangeTime = block.timestamp;
    emit PriceUpdated(nextEpoch, newPrice);
    unchecked {
        nextEpoch++;
    }
}

function exit(uint256 amount) public whenNotPaused onlyRole(MANAGER_ROLE) {
    require(balanceOf(msg.sender) >= amount, "Insufficient LP");
    uint256 ybtcAmount = convertLpToYBTC(amount); // Uses current set price
    _burn(msg.sender, amount);
    totalStaked -= ybtcAmount;
    ybtcToken.safeTransfer(msg.sender, ybtcAmount);
    emit Exit(msg.sender, amount, ybtcAmount);
}
```

User Notice Users should understand before participating in this contract:

1. LP token prices are manually set by operators, not market prices
2. Price setting includes frequency and change magnitude restrictions
3. The contract design allows prices to only increase, which may be suitable for specific business scenarios
4. Users should understand the project's price setting strategy and transparency

6. Systemic Risks

This contract adopts a centralized management model with the following systemic characteristics:

1. **Accounting Integrity Risk**: The accounting inconsistency issue in the `withdraw` function may lead to mismatches between the contract's fund records and actual state, affecting the entire system's financial integrity and users' withdrawal capabilities.

2. **Centralized Governance Structure**: The contract uses role-based access control, with administrators having emergency rescue permissions and operators controlling price setting and withdrawal approvals. This design may be suitable for scenarios requiring compliance review or centralized management.

3. **Non-Market Price Mechanism**: LP token prices are manually set by operators rather than determined through market mechanisms. Users need to understand this price setting method and its impact on investment returns.

7. Architecture and Design Observations

The contract is built upon OpenZeppelin's robust and widely-audited ERC20 and AccessControl implementations, which is a positive design choice for the base token functionality. The contract adopts a role-based access control pattern, including administrator roles (`ADMIN_ROLE`), manager roles (`MANAGER_ROLE`), and operator roles (`OPERATOR_ROLE`). This design pattern is suitable for application scenarios requiring centralized management and compliance control.

The contract's price mechanism adopts a manual setting approach, with the `setPrice` function including a `require(newPrice >= oldPrice)` constraint, ensuring prices can only increase. This design may be suitable for specific business models, such as stablecoin staking or progressive appreciation token economic models. Users should fully understand the characteristics and applicable scenarios of this price mechanism before participating.

8. Conclusion

The YBTCStakingVault contract has completed security audit. The audit identified one medium-risk accounting inconsistency issue that needs to be fixed before deployment. The contract adopts a centralized management model, including administrator emergency rescue mechanisms, operator-approved withdrawal processes, and manual price setting mechanisms. These design features are components of the contract's functionality, suitable for specific scenarios requiring centralized management and compliance control.

It is recommended to fix the accounting inconsistency issue in the `withdraw` function before deploying the contract. Users should fully understand the centralized management characteristics and operational processes before participating in this contract. It is recommended that users evaluate the team's credibility, governance transparency, and operational mechanisms.

The overall security assessment is **MEDIUM RISK** - the main risk comes from the accounting inconsistency issue, and the centralized management model requires users to fully understand its operational mechanisms. It is recommended to reassess after fixing the technical issues.

9. Disclaimer

This audit report is provided for informational purposes only and does not constitute financial advice. The analysis is based on the smart contract source code provided at a specific point in time and does not guarantee the security of the contract in perpetuity. Smart contracts are subject to inherent risks, and users should exercise extreme caution and conduct their own due diligence before interacting with