BTRToken: Security Audit Report

DogScan Security Team

DogScan

August 27th, 2025

Contents

DogScan Security Audit Report **2**
1. Executive Summary . . . 2
2. Audit Scope . . . 2
    BSC (Binance Smart Chain) . . . 2
3. Audit Methodology . . . 3
4. Findings Summary . . . 3
5. Detailed Findings . . . 4
    [I-01] Upgradeable Contract Architecture (Informational) . . . 4
    [I-02] Centralized Permission Management Structure (Informational) . . . 5
6. Systemic Risks . . . 6
7. Architecture and Design Observations . . . 7
8. Conclusion . . . 7

DogScan Security Audit Report

Project	BTRToken
Chain	BSC (ID: 56)
Proxy Contract Address	0xfed13d0c40790220fbde712987079eda1ed75c51
Implementation Contract Address	0xc8b5a0c5453c15157328b6cc1f1452be032a41f1
Audit Date	August 27th, 2025
Report Version	1.0

1. Executive Summary

The BTRToken contract has completed its security audit. This contract is an ERC20 token implemented using the OpenZeppelin Upgradeable framework, deployed with the UUPS (Universal Upgradeable Proxy Standard) pattern. The contract is deployed on BSC with minting, burning, pausing, and whitelist functionalities. The audit process found no critical, high, or medium-risk security vulnerabilities. The contract adopts role-based access control (RBAC) and an upgradeable architecture, which are technically correct and secure in their implementation.

The overall risk level is assessed as [Low Risk]. The contract is secure and reliable at the technical level, with upgradeable features and centralized governance being design choices that users should understand before participating.

2. Audit Scope

The audit scope covers the BTRToken upgradeable token contract deployed on BSC:

BSC (Binance Smart Chain)

- **Proxy Contract Address**: 0xfed13d0c40790220fbde712987079eda1ed75c51
- **Implementation Contract Address**: 0xc8b5a0c5453c15157328b6cc1f1452be032a41f1

- **Contract Type**: Upgradeable `ERC20` token contract (using UUPS proxy pattern)

The contract includes the following main components:

- **Main Contract**: `BTRToken.sol`
- **Inherited Libraries**: `OpenZeppelin Upgradeable` contracts, including `AccessControlEnumerableU`, `ERC20PermitUpgradeable`, `ERC20BurnableUpgradeable`, `PausableUpgradeable`, `UUPSUpgradeable`
- **Dependencies**: `EnumerableSetUpgradeable` utility library for whitelist management
- **Functions**: Standard `ERC20` functions, minting, burning, pausing, whitelist, and contract upgrade capabilities

3. Audit Methodology

This audit was conducted using a multi-agent AI security analysis framework. The process involves multiple specialized AI agents conducting comprehensive reviews of the smart contract's source code and on-chain state. The methodology includes:

1. **Automated Vulnerability Detection**: Static analysis agents scan code for known vulnerability patterns, logic flaws, and deviations from best practices.
2. **Upgradeability Security Analysis**: Specialized upgrade pattern agents evaluate the contract's upgrade mechanisms, storage layout compatibility, and initialization security.
3. **Access Control Review**: Agents analyze the role-based access control (RBAC) implementation, identifying privileged functions and potential centralization risks.
4. **Manual Heuristic Review**: A chief security strategist AI synthesizes all agent findings, categorizing alerts, eliminating false positives, and assessing systemic impact of combined vulnerabilities through cross-referencing call chains and contract logic.

4. Findings Summary

ID	Title	Severity	Status
I-01	Upgradeable Contract Architecture	**Informational**	**User Advisory**
I-02	Centralized Permission Management Structure	**Informational**	**User Advisory**

5. Detailed Findings

[I-01] Upgradeable Contract Architecture (Informational)

Severity: Informational

Description The contract adopts an upgradeable architecture pattern with the following technical characteristics:

1. **UUPS Proxy Pattern**: The contract uses `UUPSUpgradeable` to implement upgradeability, with separation between proxy and implementation contracts.
2. **Upgrade Authorization**: Only addresses with the `DEFAULT_ADMIN_ROLE` can authorize contract upgrades through the `_authorizeUpgrade` function.
3. **Initialization Mechanism**: Uses the `initialize` function instead of a constructor for contract initialization, and prevents direct initialization of the implementation contract through `_disableInitializers()`.
4. **Storage Safety**: Uses OpenZeppelin's upgradeable contract framework to ensure storage layout compatibility.

Impact The upgradeable architecture provides the project with the following capabilities:

- The project team can upgrade contract logic to fix bugs or add new features
- Token balances and state data are preserved during upgrades
- Requires users to trust that the project team will not perform malicious upgrades
- Upgrade authority is centralized in the contract administrator

Code Locations

```
function _authorizeUpgrade(
    address newImplementation
) internal override onlyRole(DEFAULT_ADMIN_ROLE) {}

/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}

function initialize(
    string memory name_,
    string memory symbol_,
    address owner_,
```

```
    address pauser_,
    uint256 mintQuota_
) public initializer { ... }
```

User Advisory Users should understand before participating in this token:

1. This is an upgradeable token contract where the contract logic may change in the future
2. Upgrade authority is controlled by the contract administrator, requiring user trust that no malicious upgrades will occur
3. It is recommended to monitor the project's upgrade governance mechanisms and multi-signature management
4. Contract upgrades may change the behavioral characteristics of the token

[I-02] Centralized Permission Management Structure (Informational)

Severity: Informational

Description The contract adopts a centralized role-based permission management model with the following roles and permissions:

1. **DEFAULT_ADMIN_ROLE Permissions**:
   - Can upgrade contract implementation
   - Can set mint quota (`setMintQuota`)
   - Can assign and revoke minter roles (`setMinter`)
   - Can assign and revoke burner roles (`setBurner`)
2. **MINTER_ROLE Permissions**:
   - Can mint new tokens within quota limits (`mint`)
3. **BURNER_ROLE Permissions**:
   - Can burn tokens from any address (`burn`)
4. **PAUSER_ROLE Permissions**:
   - Can pause and unpause token transfers (`pause`/`unpause`)
   - Can manage whitelist (`setWhitelister`)

Impact Under this permission structure, the contract has the following characteristics:

- The project team has complete control over token supply
- Can pause token transfers (except for whitelisted addresses)
- Has the ability to burn any user's tokens
- Has the ability to mint new tokens (subject to quota limits)
- All critical operations are controlled by centralized roles

Code Locations

```
bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

function setMinter(address minter, bool enabled) public onlyRole(
    DEFAULT_ADMIN_ROLE) { ... }
function setBurner(address burner, bool enabled) public onlyRole(
    DEFAULT_ADMIN_ROLE) { ... }
function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE)
    { ... }
function burn(address account, uint256 amount) public onlyRole(
    BURNER_ROLE) { ... }
function pause() external onlyRole(PAUSER_ROLE) whenNotPaused { ... }
```

User Advisory Users should understand before participating in this token:

1. This is a centrally managed token contract where the project team has extensive control permissions
2. Token supply may change through minting and burning functions
3. The project team can pause token transfers and manage whitelists
4. Users should evaluate the team's credibility and governance transparency
5. It is recommended to monitor the project's multi-signature management and permission allocation mechanisms

6. Systemic Risks

After audit, this contract exhibits the following systemic characteristics:

1. **High Technical Security**: The contract is based on the mature `OpenZeppelin Upgradeable` framework and is secure and reliable at the technical level. All core functions (transfers, approvals, pausing, etc.) strictly follow standard implementations.

2. **Upgradeable Architecture**: Uses the `UUPS` proxy pattern, providing upgrade flexibility for the project while introducing trust requirements related to upgrades. The upgrade mechanism is technically secure in its implementation.
3. **Centralized Governance Structure**: The contract uses role-based access control where administrators have extensive control permissions, including upgrade, minting, burning, and pausing functions. This design provides operational flexibility for the project but also means centralized control exists.
4. **Standard Compatibility**: The contract is fully compatible with the `ERC20` standard and implements the `ERC20Permit` extension, providing gas-free authorization functionality.

7. Architecture and Design Observations

The contract is built on a solid foundation using `OpenZeppelin Upgradeable` standard, secure contract implementations. The architecture is clear and follows industry best practices for upgradeable contracts:

1. **Proxy Pattern Implementation**: Correctly uses the `UUPS` proxy pattern, with storage variable definitions following upgrade safety specifications, avoiding storage layout conflicts.
2. **Role Management System**: The contract uses a role management system based on `AccessControlEnumerableUpgradeable`, allowing efficient management of multiple privileged roles. This design provides flexibility while maintaining code clarity.
3. **Initialization Security**: Properly implements the initialization mechanism, ensuring single initialization through the `initializer` modifier and disabling implementation contract initialization in the constructor.
4. **Pause and Whitelist Mechanism**: Implements fine-grained transfer control, allowing minting, burning, and whitelisted address transfers even in paused state.

8. Conclusion

The `BTRToken` contract is a well-implemented upgradeable `ERC20` token using the thoroughly tested `OpenZeppelin Upgradeable` framework. The contract includes upgradeable architecture and centralized permission management structure, which are technically correct and secure in implementation. While upgradeable and centralized features exist, these are design choices and are well-implemented.

Safe for deployment and use: No security risks were found at the technical level, and all functions are correctly implemented. Users should fully understand the upgradeable features and centralized

governance structure before participating in this token. It is recommended that users evaluate the project team's credibility, governance transparency, and upgrade governance mechanisms.

Overall security assessment: [Low Risk] - Technical implementation is secure and reliable, upgradeability and centralized features are design choices, users need to fully understand the governance mechanisms and upgrade permissions.