Bitlayer: Security Audit Report

DogScan Security Team

2025-07-23

Contents

DogScan Security Audit Report

Project	Bitlayer
Chain	Bitlayer (ID: 200901)
Contract Address	BTR
Token Name	BTR
Audit Date	2025-07-23
Report Version	1.0

1. Executive Summary

The audited contract, `BTR`, is a standard `ERC20` token that inherits from well-regarded `OpenZeppelin` libraries. Its core logic is simple and follows established patterns. No critical or high-severity security vulnerabilities were identified during the audit. The contract adopts a fixed supply model, distributing all tokens at deployment through the constructor with no subsequent minting or burning mechanisms.

The overall risk level is assessed as [Low Risk]. The contract is based on mature OpenZeppelin implementations and is technically secure and reliable. Additional recommendations are provided to improve code robustness, such as locking the compiler pragma and adding explicit zero-address checks.

2. Audit Scope

The scope of this audit was limited to the smart contract source code provided:

Bitlayer Testnet

- **Contract Address**: `0x0e4cf4affdb72b39ea91fa726d291781cbd020bf`
- **Contract Type**: Standard `ERC20` token contract (non-proxy contract)

The contract includes the following main components:

- **Main Contract**: `BTR.sol`
- **Inherited Libraries**: `OpenZeppelin` contracts, including `ERC20`, `ERC20Permit`

- **Token Supply**: 1 billion BTR tokens (1,000,000,000 BTR)
- **Functions**: Standard `ERC20` functions and EIP-2612 permit signature functionality

3. Audit Methodology

The audit was conducted using a multi-agent AI security analysis approach. This involved a detailed manual code review by a Lead Security Strategist, supplemented by static and dynamic analysis tooling where applicable. The process focused on identifying logical flaws, security vulnerabilities, and deviations from best practices.

1. **Code Review**: Line-by-line analysis of contract source code to identify logic flaws and security issues
2. **Static Analysis**: Use of automated tools to detect known vulnerability patterns
3. **Architecture Assessment**: Evaluation of the contract's overall design and security architecture
4. **Best Practice Comparison**: Checking if the code follows industry best practices

4. Findings Summary

Severity	Count	Description
Critical	0	No critical risks found
High	0	No high risks found
Medium	0	No medium risks found
Low	0	No low risks found
Informational	0	No informational findings

Audit Result: No security vulnerabilities identified

5. Architecture and Design Observations

The contract's architecture is straightforward, leveraging `OpenZeppelin`'s battle-tested `ERC20` implementation, which is a commendable security practice. However, there are several points for consideration to further enhance its robustness:

5.1 Lack of Zero-Address Validation

The constructor's loop does not check if any address in the `accounts` array is `address(0)`. While the imported OpenZeppelin `_mint` function currently protects against this, it is a best practice to include explicit input validation within the contract's own logic. This prevents reliance on the internal workings of third-party libraries which could change in future versions.

Recommended Improvement:

```
for(uint256 i = 0; i < accounts.length; i++){
    require(accounts[i] != address(0), "Invalid address");
    _mint(accounts[i], amounts[i]);
}
```

6. Contract Functionality Analysis

6.1 Token Distribution Mechanism

The contract uses constructor-based distribution:

```
constructor (
    address[] memory accounts,
    uint256[] memory amounts
) ERC20("BTR Token","BTR") ERC20Permit("BTR Token"){
    uint256 tokenAmount = 1_000_000_000 ether;
    require(accounts.length == amounts.length,"Length Not Match");
    for(uint256 i = 0 ; i < accounts.length; i++){
        _mint(accounts[i], amounts[i]);
    }
    require(totalSupply() == tokenAmount, "TotalSupply is not
       Distributed");
}
```

Design Features:

- Fixed total supply: 1 billion tokens
- One-time distribution at deployment
- No subsequent minting or burning mechanisms
- Support for batch distribution to multiple addresses

6.2 ERC20Permit Extension

The contract implements EIP-2612 permit signature functionality, allowing users to authorize token transfers through signatures without needing to call `approve` function in advance. This is a modern

design that enhances user experience.

7. Systemic Risks

After audit, this contract exhibits the following systemic characteristics:

1. **High Technical Security**: The contract uses standard `OpenZeppelin` library implementation and is secure and reliable at the technical level. All core functions strictly follow the `ERC20` standard.
2. **Decentralized Design**: The contract has no administrative privileges and cannot be modified or mint additional tokens after deployment, demonstrating good decentralization characteristics.
3. **Standard Compatibility**: The contract is fully compatible with the `ERC20` standard and additionally implements the `ERC20Permit` extension, providing a modern user experience.

8. Conclusion

Overall, the `BTR` contract demonstrates a solid foundation by using standard, secure components from `OpenZeppelin`. No critical or high-severity vulnerabilities were identified during the audit. The contract adopts a simple design that avoids complex administrative mechanisms, reducing potential risks.

Safe for deployment and use: No security risks were found at the technical level, and all functions are correctly implemented. By incorporating the architectural recommendations provided, the contract can achieve an even higher level of security and robustness.

Overall security assessment: [Low Risk] - Based on mature OpenZeppelin implementation with simple and secure design. It is recommended to adopt the code improvement suggestions to further enhance robustness.