
# Bitgold Audit Report

September 2025

# Bitgold Audit Report

Zero Drift

September 2025

Prepared by: Zero Drift

## Table of Contents



## About Zero Drift

Zero Drift is a tech-driven Web3 auditing firm dedicated to zero-incident smart-contract assurance for mission-critical protocols.

Our team consists of several senior smart-contract auditors, each with 5+ years of experience and a perfect post-audit record across more than 80 successful reviews, collectively helping protect over $5 billion in on-chain assets.

## Disclaimer

This report presents the Bitgold audit in Zero Drift's reporting format.

As with any time-boxed smart contract assessment, this document should not be treated as an endorsement of the project, business model, or token economics. The review is limited to the code and behaviors captured within the assessed contract scope.

## Risk Classification

| | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## Protocol Overview

Bitgold is a token contract implementation deployed on BNB Smart Chain, with the reviewed logic centered around ERC-20 style allowance and delegated transfer flows such as `approve()` and `transferFrom()`.

The review focused on allowance management behavior, event emission, and input validation around spender authorization. The audited contract scope was primarily concerned with token approval safety, off-chain allowance observability, and user protection against invalid authorization targets.

## Audit Scope

The scope consisted of a focused review of the deployed Bitgold token contract implementation.

- Project name: Bitgold
- Audit timeline: 2025-09-22 to 2025-09-24
- Language: Solidity
- Repository / deployment reference: https://bscscan.com/address/0x4c9027e10c5271efca82379d3123917ae3f2374e

- Commit hash: N/A
- Files reviewed: `BitgoldContract.sol`

## Executive Summary

Over the course of the review, three issues were identified in the Bitgold contract: one High severity issue and two Low severity issues. Two findings were marked Fixed and one Low severity issue was marked Acknowledged.

The most significant issue involved the standard allowance race condition in `approve()`, where overwriting a non-zero allowance can allow spending against both the old and new allowance in certain transaction ordering scenarios. Additional issues covered missing `Approval` event emission after allowance reductions in `transferFrom()` and missing zero-address validation for `approve()` spender inputs.

### Summary

| Project Name | Bitgold |
|---|---|
| Repository | BscScan Address |
| Commit | N/A |
| Audit Timeline | Sep 22-24, 2025 |
| Methods | Static Analysis, Manual Review |

### Issues Found

| | Count |
|---|---|
| Critical Risk | 0 |
| High Risk | 1 |
| Medium Risk | 0 |
| Low Risk | 2 |
| Informational | 0 |

| | Count |
|---|---|
| Gas Optimizations | 0 |
| **Total Issues** | **3** |

**Summary of Findings**

| ID | Title | Severity | Status |
|---|---|---|---|
| H-1 | `approve()` race condition | High | Resolved |
| L-1 | Missing `Approval` event update in `transferFrom()` | Low | Resolved |
| L-2 | Missing zero-address check in `approve()` | Low | Acknowledged |

# Findings

## High Risk

### [H-01] `approve()` Race Condition

Description

`approve()` directly overwrites the previous allowance amount. This creates the well-known ERC-20 approval race condition: if a spender uses the old allowance while the owner is attempting to update it, the spender may be able to consume both the prior allowance and the replacement allowance depending on transaction ordering.

Impact

Unexpected overuse of authorized funds may occur if allowance changes are front-run or otherwise reordered around spender activity.

Recommendation

Adopt safer allowance mutation patterns such as `increaseAllowance()` and `decreaseAllowance()`, or enforce the common mitigation of only allowing a new non-zero allowance when the current allowance is zero.

Status: Resolved

## Low Risk

### [L-01] Missing `Approval` Event Update in `transferFrom()`

Description

After `transferFrom()` reduces a spender's allowance, no `Approval` event is emitted to reflect the updated allowance. This can create divergence between on-chain state and off-chain indexers or monitoring systems that rely on event streams for allowance tracking.

Impact

Allowance changes become harder to monitor accurately off-chain, which may affect explorers, dashboards, and internal compliance or analytics tooling.

Recommendation

Emit an `Approval(sender, spender, newAllowance)` event immediately after the allowance is reduced in `transferFrom()`.

Status: Resolved

### [L-02] Missing Zero-Address Check in `approve()`

Description

`approve()` allows the caller to set the spender to the zero address. While this does not create a direct theft vector, it can lead to misconfiguration and degraded user experience if approvals are mistakenly granted to an unusable address.

Impact

Users may accidentally assign allowance to the zero address, resulting in avoidable mistakes and inconsistent validation behavior.

Recommendation

Add explicit input validation to reject `spender == address(0)` in `approve()` and any related allowance-adjustment helpers.

Status: Acknowledged