0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Brilliant Coin

August 2025

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Brilliant Coin**.

The code is available at the @bebrilliant1/brilliantcoin GitHub repository and was audited after the commit 7a501b2a8db8170644367ad5d688ea1251470e72. A recheck was done after the commit c896e961a89bd6cff750910f13f41b2fa247d0f8

The scope of the audit is the following smart contracts:

- BrilliantOracle.sol

- BrilliantReferralManager.sol

- BrilliantStaking.sol

Name	Brilliant Coin
Audit date	2025-08-19 - 2025-08-20
Language	Solidity
Platform	Base Chain

Contracts checked

Name	Address
BrilliantOracle	
BrilliantStaking	
BrilliantReferralManager	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed

Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.

Medium severity Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.

Low severity Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. The owner can mint arbitrary number of Brilliant tokes (BrilliantStaking)

Status: Acknowledged

The BrilliantStaking contract allows the owner to set an arbitrary `referralManager` address via the `setReferralManager()` function. The `claimReferralRewards()` function then calls `referralManager.claimReferralRewards(msg.sender)` to determine the amount of BrilliantToken to mint for the caller. Crucially, the BrilliantStaking contract blindly trusts the uint256 value returned by the external `referralManager.claimReferralRewards()` call and uses this value directly in `brilliantToken.mint(msg.sender, pending)`.

```
    function claimRewards() external payable nonReentrant {
        UserInfo storage user = userInfo[msg.sender];

        // Update pool
        updateAccRewards();

        // Calculate pending rewards
        uint256 pending = ((user.amount + user.vestedAmount) * accBrilliantPerShare /
ACC_PRECISION) - user.rewardDebt;
        pending += user.pendingRewards;

        // Check if rewards exceed minimum threshold
        require(pending >= minClaimThreshold, "GenStaking: Claim amount below min");

        // Calculate and distribute referral rewards
        if (pending > 0) {
            // Process the fee
            uint256 refundAmount = _checkAndProcessFee(pending);
```

```
            // Refund any excess fee
            if (refundAmount > 0) {
                (bool refundSuccess,) = msg.sender.call{ value: refundAmount }("");
                require(refundSuccess, "GenStaking: Refund failed");
            }

            user.pendingRewards = 0;
            user.rewardDebt = (user.amount + user.vestedAmount) *
accBrilliantPerShare / ACC_PRECISION;

            // Mint BRILL tokens as rewards
            brilliantToken.mint(msg.sender, pending);

            // Process referral rewards if referral manager is set
            if (address(referralManager) != address(0)) {
                referralManager.processRewardClaim(msg.sender, pending);
            }

            emit RewardClaimed(msg.sender, pending);
        }
    }
```

By setting a malicious referralManager smart contract the owner can mint any number of Brilliant tokens to a specified address.

Since the referralManager can only be set once, there is no risk of manipulation after it has been assigned to the correct contract address.

Medium severity issues

1. Reward claims can be blocked by the contract owner (BrilliantStaking)

Status: Acknowledged

The contract allows the owner to set a referralManager address via the `setReferralManager()` function.

The

`claimRewards()` function attempts to process referral rewards by calling referralManager.processRewardClaim(msg.sender, pending) if address of referral manager contract is not a zero address. Setting it, for example, for an EOA address will effectively block all reward claims.**Since the referralManager can only be set once, there is no risk of manipulation after it has been assigned to the correct contract address.**

Low severity issues

1. Misleading errors (BrilliantStaking)

Status: Fixed

The error messages in the BrilliantStaking contract are misleading as they incorrectly use the prefix "GenStaking" instead of "BrilliantStaking".

Conclusion

Brilliant Coin BrilliantOracle, BrilliantStaking, BrilliantReferralManager contracts were audited. 1 high, 1 medium, 1 low severity issues were found.
1 low severity issue has been fixed in the update.

Privileged roles have extensive control over the protocol's core functions and funds. To address this, we recommend implementing robust security practices, including the use of multi-signature wallets, timelocks, and hardware wallets for priviledged accounts management. Users are required to trust that the administrators that these accounts are properly secured and will not act maliciously.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

0x Guard