SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Bolide
Date: July 7th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Bolide.
Approved By	Andrew Matiukhin | CTO Hacken OU
Type of Contracts	ERC20 token; Farming; TokenSale; Strategy; Vesting
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://bolide.fi/
Timeline	21.03.2022 - 07.07.2022
Changelog	30.03.2022 - Initial Review 18.04.2022 - Revise 07.06.2022 - Revise 07.07.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Bolide (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/bolide-fi/contracts
Commit:
57f192ff4c4f1c8e8fbb99c60757f0327d76716c
Documentation: Yes
JS tests: Yes
Contracts:
token/contracts/Bolide.sol
token/contracts/TokenVesting.sol
token/contracts/VestingController.sol
vesting/contracts/TreasuryVester.sol
vesting/contracts/libs/Bolide.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided some functional requirements and no technical requirements. The total Documentation Quality score is **8** out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Not following solidity code style guidelines.

Architecture quality

The architecture quality score is **10** out of **10**.

Security score

As a result of the audit, security engineers found **2** low severity issues. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

1. **No tests**

   It is highly recommended to add tests for the contract's code. Unit tests would help ensure functions are working properly, while integration tests would ensure contracts are working perfectly.

   Contracts: Bolide.sol, TreasuryVester.sol

   Recommendation: Create unit and integration tests covering up to 100% statements and branches.

   Status: Fixed (Revised Commit: 9ca0cf0)

🟦 Low

1. **Floating solidity version**

   It is recommended to specify the exact solidity version in the contracts.

   Contracts: all

   Recommendation: Specify the exact solidity version (ex. pragma solidity 0.8.10 instead of pragma solidity ^0.8.0).

   Status: Fixed (Revised Commit: 9378f79)

2. **Outdated solidity version**

   It is not recommended to use an outdated solidity version.

   Contracts: TreasuryVester.sol

   Recommendation: Do not use solidity 0.6.12, use 0.8.10-0.8.13 instead.

   Status: Fixed (Revised Commit: 9378f79)

3. **Implicit variables visibility**

   State variables that do not have specified visibility are declared **internal** implicitly. That could not be obvious.

   Contract: Bolide.sol

   Variable: timestampCreated

   Recommendation: Always declare visibility explicitly.

Status: Reported (Revised Commit: 9ca0cf0)

4. **A public function that could be declared external**

Public functions that are never called by the contract should be declared **external**.

Contracts: Bolide.sol, TreasuryVester.sol

Functions: Bolide.name, Bolide.symbol, Bolide.decimals, Bolide.totalSupply, Bolide.balanceOf, Bolide.cap, Bolide.transfer, Bolide.approve, Bolide.burn, Bolide.burnFrom, Bolide.transferFrom, Bolide.increaseAllowance, Bolide.decreaseAllowance, TreasuryVester.setRecipient, TreasuryVester.claim

Recommendation: Use the **external** attribute for functions never called from the contract.

Status: Reported (Revised Commit: 9ca0cf0)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.