HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BullRun Finance
Date: June 25th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for BullRun Finance
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Governance, Locker, Masterchef, NFT, ERC20, Rewards
Platform	BSC / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git Repository	https://github.com/BullRunFinance/contracts/tree/9d9ced16b56f977c17afee231e745c7a9e55c84e
Timeline	9 June 2021 - 22 June 2021
Changelog	22 June 2021 - Initial Audit 25 June 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by BullRun Finance (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between June 9th, 2021 - June 22th, 2021. The second code review conducted on June 25th, 2021.

Scope

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

Repository:
https://github.com/BullRunFinance/contracts/tree/9d9ced16b56f977c17afee231e745c7a9e55c84e

Files:
- BullGovernance.sol
- BullLocker.sol
- BullMasterchef.sol
- BullNFT.sol
- BullReferral.sol
- BullToken.sol
- BullVoteProxy.sol
- Migrations.sol
- RewardDistribution.sol
- Timelock.sol

We have scanned these smart contracts 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 ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call

	▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset’s integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contract is secured.

Insecure	Poor secured	Secured	Well-secured

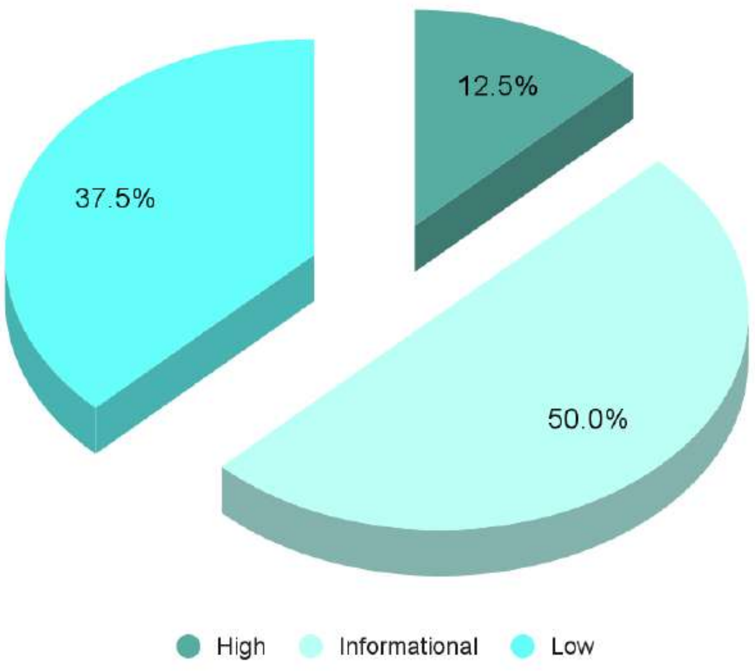
You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

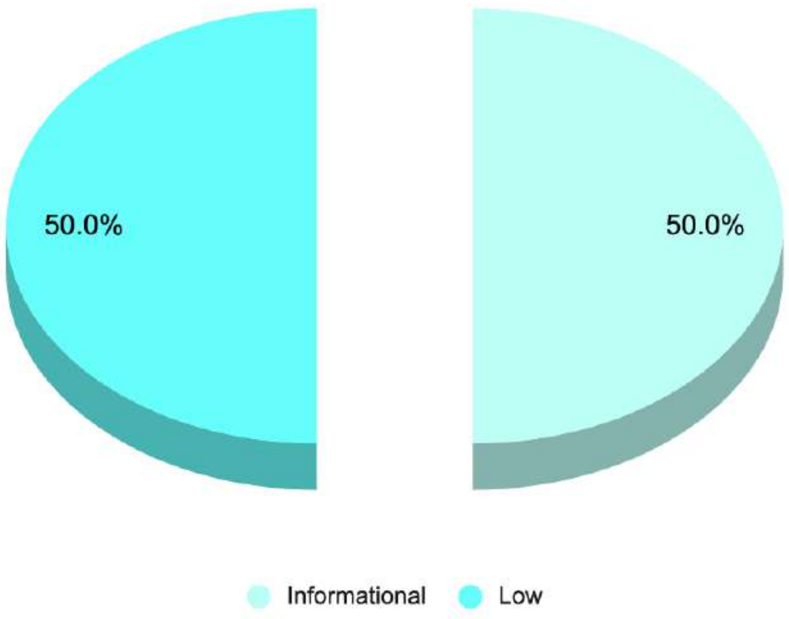
Security engineers found **1** high, **3** low and **4** informational issues during the first review.

Security engineers found **1** low and **1** informational issues during the second review.

Graph 1. The distribution of vulnerabilities after the first review.



Graph 2. The distribution of vulnerabilities after the second review.



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 can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit overview

■ ■ ■ ■ Critical

No Critical severity issues were found.

■ ■ ■ High

Vulnerability: Uninitialized state variable

State variable startBlock never initialized but used in the comparison.

Recommendation: please initialize the startBlock variable when starting the contract execution.

Fixed before the second review.

■ ■ Medium

No Medium severity issues were found.

■ Low

1. **Vulnerability**: Costly loops

   _massUpdatePools may run out-of-gas if too many tokens are added. This may have implications on calling functions **add** and **set**.

   Lines: BullMasterchef.sol#196-201

```
function _massUpdatePools() private {
   uint256 length = poolInfo.length;
   for (uint256 pid = 0; pid < length; ++pid) {
       _updatePool(pid);
   }
}
```

2. **Vulnerability**: State variable should be constant

   State variables which never change their values should be declared constant to save gas.

   Fixed before the second review.

3. **Vulnerability**: State variable should be immutable

State variables which set their values in the constructor and never change their values after that should be declared immutable to save gas.

Fixed before the second review.

Lowest / Code style / Best Practice

1. **Vulnerability**: Constructor visibility

   In Solidity versions greater or equal to 0.7.0 , the visibility for the constructor can be omitted.

   Recommendation: Please consider removing explicit visibility from the **constructor**

   Fixed before the second review.

2. **Vulnerability:** Public function that could be declared external

   public functions that are never called by the contract should be declared **external** to save gas.

   Fixed before the second review.

3. **Vulnerability:** Boolean equality

   Boolean constants can be used directly and do not need to be compared to true or false.

   Fixed before the second review.

4. Lines

- 33, 36, 42, 224 and 226 of the BullGovernance.sol
- 105, 142, 165, 182, 217, 226, 307 and 440 of the BullMasterchef.sol
- 36 of the BullNFT.sol
- 41 of the BullReferral.sol
- 114, 127, 134-139, 147, 164, 257, 267, 277, 322 and 355 of the BullToken.sol
- 85 and 87 of the BullVoteProxy.sol
- 131, 132, 148, 149 and 210 of the RewardDistribution.sol
- 25-27, 42, 43, 56, 63, 73, 75, 83, 85, 94, 103 and 107-109 of the Timelock.sol

are above the recommended maximum line length.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **1** high, **3** low and **4** informational issues during the first review.

Security engineers found **1** low and **1** informational issues during the second review.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

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