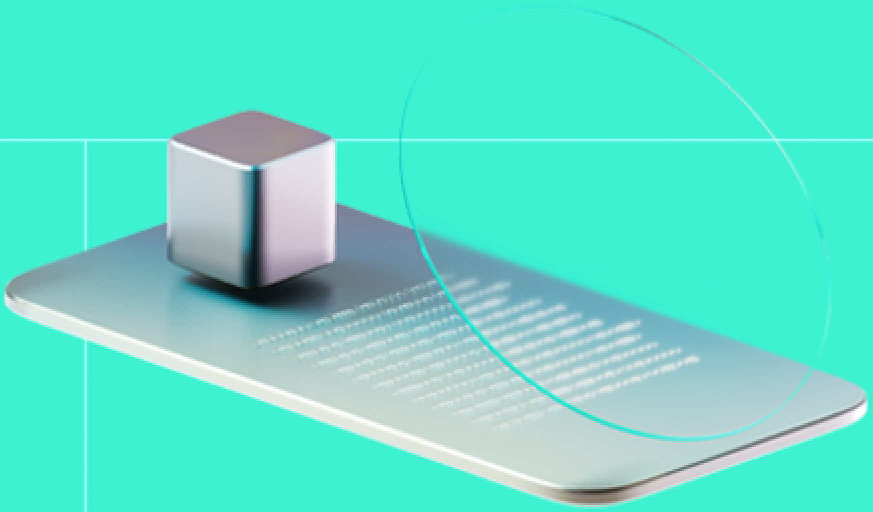
Smart Contract Code Review And Security Analysis Report



Customer: BlastUp

Date: 10/05/2024

We express our gratitude to the BlastUp team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

BlastUP is a launchpad and staking platform within the Blast blockchain ecosystem.

Platform: Blast(EVM)

Language: Solidity

Tags: Blast, Staking

Timeline: 06/05/2024 - 07/05/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/blastupio/launchpad-contracts
Commit	cb6957d

Audit Summary

10/10	10/10	100%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	1
Medium	0
Low	0

Vulnerability	Status
F-2024-2196 - Insufficient Reward Leads To Denial Of Service	Fixed

Document

Name	Smart Contract Code Review and Security Analysis Report for BlastUp
Audited By	Eren Gonen
Approved By	Ataberk Yavuzer
Website	https://blastup.io/
Changelog	08/05/2024 - Preliminary Report
	10/05/2024 - Final Report

Table of Contents

System Overview

BlastUP is a staking protocol with the following contracts:

BLPStaking — a contract that rewards users for staking their tokens. Users can earn token based on the percentage which is set by admin

Privileged roles

- The owner of the BLPStaking contract can set the minimum staking amount, lock time, and percentage.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage)

Security score

Upon auditing, the code was found to contain **0** critical, **1** high, **0** medium, and **0** low severity issues. Out of these, **1** issues have been addressed and resolved, leading to a security score of **10** out of **10**.

All identified issues are detailed in the "Findings" section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score **10** out of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- If there are not enough rewards in the contract, users may not be able to claim their rewards or may be forced to use emergency withdrawal without claiming their rewards.

Findings

Vulnerability Details

F-2024-2196 - Insufficient Reward Leads To Denial Of Service - High

Description:

The **BLPStaking** contract facilitates staking operations where users stake tokens and earn rewards over time based on the staking parameters set by the administrator. A critical oversight in the contract design is the lack of separation between the tokens used for staking and those designated for rewards. Since both operations utilize the same token address without distinct accounting for the reward pool, users can potentially stake tokens even when there are insufficient funds in the contract to cover future rewards. This scenario can lead to a temporary Denial of Service (DOS) in the `claim()` and `withdraw()` functions, as users may not be able to withdraw or claim rewards unless additional funds are deposited into the contract by the administrator.

In the contract, both staking deposits and reward payouts utilize the `stakeToken`. If the contract allows withdrawals or claims despite insufficient dedicated reward funding, users may inadvertently receive tokens from the staking pool itself—i.e., other users' deposits. This can deplete the staking pool, potentially leading to a shortfall when other users attempt to claim rewards or withdraw their stakes. Due to the lack of a dedicated reward reserve, if the contract's balance falls below the sum required to cover all potential reward claims, users might be unable to withdraw their stakes or claim rewards, leading to a temporary Denial of Service (DOS).

Assets:

- BLPStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Fixed

Classification

Impact:	4/5
Likelihood:	4/5
Exploitability:	Independent
Complexity:	Medium

Likelihood [1-5]: 4

Impact [1-5]: 4

Exploitability [0-2]: 0

Complexity [0-2]: 1

Final Score: 3.8 (High)

Hacken Calculator Version: 0.6

Severity: High

Recommendations

Remediation: To enhance the functionality and security of the system, we advise implementing the following measures:

Separate Reward Reserve:

- Implement a dedicated reward reserve to ensure that rewards are disbursed from a separate pool of funds, not from the staked tokens. This can be achieved by maintaining a distinct **reward reserve** balance in the contract.

Funding Checks:

- Introduce stringent checks to prevent new stakes unless there are sufficient funds to cover both current and future rewards.

Emergency Withdrawal Logic(Optional):

- Consider implementing emergency withdrawal logic as an additional layer of security. This logic would prioritize user withdrawals over reward payouts in cases where funds are limited. While optional, this feature can provide added assurance to users during unforeseen circumstances.

Resolution: The BlastUP team fixed the issue in commit **7b1cddc** by integrating a force withdrawal mechanism and an `ensureSolvency()` modifier into the `claim()` and `withdrawFunds()` functions. This modifier guarantees that the contract's balance exceeds the total amount owed to users, not including rewards.

Evidences

POS

Reproduce: The Proof of Concept (POC) scenario described in the Solidity testing contract **StakingEren** can be broken down into five key steps to illustrate the issues and potential risks with the **BLPStaking** contract as follows:

Step 1: Setting Up and Staking Tokens

- The test simulates a user who is staking a certain amount of tokens (**1e18** wei, or 1 token in this case) for a lock period of 1 month (**2592000** seconds).
- The user is first minted tokens to have enough balance to stake.
- The admin sets the reward percentage for this lock period to 100% per annum.
- The user approves the staking contract to spend their tokens and then stakes them, expecting an emission of the **Staked** event.

Step 2: First Claim After One Day

- The blockchain time is moved forward by 1 day (**86400** seconds).
- Before the user claims their rewards, their token balance is checked.
- The test checks if the reward calculated for one day is as expected (**27397260273972** wei, or approximately **0.027397** tokens). This is derived from the 100% annual reward rate, pro-rated for one day.
- The user then claims their reward, and the test verifies that the user's token balance has increased exactly by the amount of the daily reward.

Step 3: Time Warp to End of Lock Period

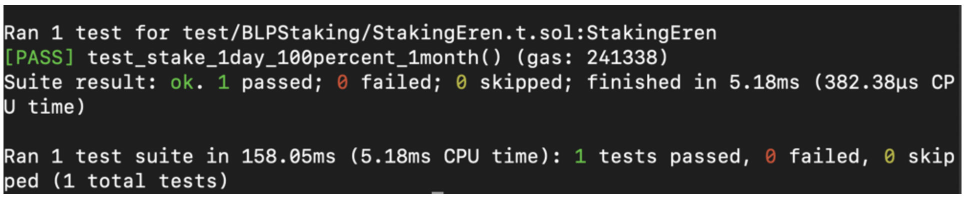
- The blockchain time is fast-forwarded to the end of the lock period (1 month).

Step 4: Attempted Withdrawal

- The user attempts to withdraw their staked tokens and the earned rewards.
- The scenario is set up to expect a revert, indicating that the withdrawal is not successful. This part of the test is designed to demonstrate a possible failure point in the contract, likely due to insufficient tokens available in the contract to cover both the principal and the accrued rewards, as previously discussed.

This scenario effectively demonstrates the critical design flaw where insufficient funds due to overlapping staking and reward distributions can lead to failures in user withdrawals, reinforcing the need for separate management of staked funds and rewards.

Results:



```
Ran 1 test for test/BLPStaking/StakingEren.t.sol:StakingEren
[PASS] test_stake_1day_100percent_1month() (gas: 241338)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.18ms (382.38µs CP
U time)

Ran 1 test suite in 158.05ms (5.18ms CPU time): 1 tests passed, 0 failed, 0 skip
ped (1 total tests)
```

Files: DosPoc.t.sol

Observation Details

F-2024-2200 - Missing Zero Address Validation - Info

Description: In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Missing checks were observed in the following functions of the `BLPStaking` contract:

- `./BLPStaking.sol: constructor()`

Assets:

- BLPStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Recommendations

Remediation: Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Resolution: This finding was acknowledged by the BlastUP team.

F-2024-2201 - Missing Events - Info

Description:	Events for critical state changes should be emitted for tracking actions off-chain.
	
	It was observed that events are missing in the following functions:
	
	`setMinBalance()`
	`setLockTimeToPercent()`
	
	Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:	• BLPStaking.sol [https://github.com/blastupio/launchpad-contracts]
Status:	Accepted

Recommendations

Remediation:	Consider implementing and emitting events for the necessary functions.
Resolution:	This finding was acknowledged by the BlastUP team.

F-2024-2202 - Use `Ownable2Step` rather than `Ownable - Info

Description: Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
contract BLPStaking is Ownable {...}
```

Assets:

- BLPStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Recommendations

Remediation: Consider using `Ownable2Step` or `Ownable2StepUpgradeable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Resolution: This finding was acknowledged by the BlastUP team.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/blastupio/launchpad-contracts
Commit	cb6957dde5944e6cfd885ceed8539140dce51a98
Whitepaper	N/A
Requirements	https://docs.blastup.io/blastup-docs
Technical Requirements	Confidential

Contracts in Scope

./contracts/BLPStaking.sol