Smart Contract Audit Report

for

Bondly.Finance

TRUSTLOOK

Version 1.0

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	Bondly.Finance
Contract codebase	N/A
Platform	Ethereum
Language	Solidity
Submission Time	2021.09.16

Report Overview

Report ID	TBL_20210916_00
Version	1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.09.19
Finished Time	2021.09.30

Disclaimer

Trustlook audit reports do not provide any warranties or guarantees on the vulnerability-free nature of the given smart contracts, nor do they provide any indication of legal compliance. The Trustlook audit process is aiming to reduce the high level risks possibly implemented in the smart contracts before the issuance of audit reports. Trustlook audit reports can be used to improve the code quality of smart contracts and are not able to detect any security issues of smart contracts that will occur in the future. Trustlook audit reports should not be considered as financial investment advice.

About Trustlook Blockchain Labs

Trustlook Blockchain Labs is a leading blockchain security team with a goal of security and vulnerability research on current blockchain ecosystems by offering industry-leading smart contracts auditing services. Please contact us for more information at (https://www.trustlook.com/services/smart.html) or Email (bd@trustlook.com)

The Trustlook blockchain laboratory has established a complete system test environment and methods.

Black-box Testing	The tester has no knowledge of the system being attacked. The goal is to simulate an external hacking or cyber warfare attack.
White-box Testing	Based on the level of the source code, test the control flow, data flow, nodes, SDK etc. Try to find out the vulnerabilities and bugs.
Gray-box Testing	Use Trustlook customized script tools to do the security testing of code modules, search for the defects if any due to improper structure or improper usage of applications.

Introduction

By reviewing the implementation of Bondly.Finance's smart contracts, this audit report has been prepared to discover potential issues and vulnerabilities of their source code. We outline in the report about our approach to evaluate the potential security risks. Advice to further improve the quality of security or performance is also given in the report.

About Bondly.Finance

Bondly.Finance is a data aggregator specially designed for DeFi for real world assets, providing customisable smart contracts to simplify the investment process of DeFi for users of all levels.

About Methodology

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contracts related security issues using automatic verification tools and manual review. To discover potential logic weaknesses or project specific implementations, we thoroughly discussed with the team to understand the business model and reduce the risk of unknown vulnerabilities. For any discovered issue, we might test it on our private network to reproduce the issue to prove our findings.

The checklist of items is shown in the following table:

Category	Type ID	Name	Description
Coding Specification	CS-01	ERC standards	The contract is using ERC standards.
	CS-02	Compiler Version	The compiler version should be specified.
	CS-03	Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
	CS-04	Return standard	Following the ERC20 specification, the transfer and approve

			functions should return a bool value, and a return value code needs to be added.
	CS-05	Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors.
	CS-06	Unused Variable	Unused variables should be removed.
	CS-07	Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
	CS-08	Event Standard	Define and use Event appropriately
	CS-09	Safe Transfer	Using transfer to send funds instead of send.
	CS-10	Gas consumption	Optimize the code for better gas consumption.
	CS-11	Deprecated uses	Avoid using deprecated functions.
	CS-12	Sanity Checks	Sanity checks when setting key parameters in the system
Coding Security	SE-01	Integer overflows	Integer overflow or underflow issues.
	SE-02	Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
	SE-03	Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
	SE-04	Tx.origin usage	Avoid using tx.origin for authentication.
	SE-05	Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
	SE-06	Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
	SE-07	External call checks	For external contracts, pull instead of push is preferred.
	SE-08	Weak random	The method of generating random numbers on smart contracts requires more considerations.
Additional Security	AS-01	Access control	Well defined access control for functions.
	AS-02	Authentication management	The authentication management is well defined.
	AS-03	Semantic Consistency	Semantics are consistent.
	AS-04	Functionality checks	The functionality is well implemented.

	AS-05	Business logic review	The business model logic is implemented correctly.

The severity level of the issues are described in the following table:

Severity	**Description**
Critical	The issue will result in asset loss or data manipulations.
High	The issue will seriously affect the correctness of the business model.
Medium	The issue is still important to fix but not practical to exploit.
Low	The issue is mostly related to outedate, unused code snippets.
Informational	This issue is mostly related to code style, informational statements and is not mandatory to be fixed.

Audit Results

Here are the audit results of the smart contracts. The new release of the smart contracts add more features to restrict the privilege of the owner to reduce the risk of private key loss or hacking events.

Scope

Following files have been scanned by our internal audit tool and manually reviewed and tested by our team:

File names	Sha1
BondlyStaking.sol	8e95a3e9714fa28c1f76bfa2a16ddace9ce1ce1d
LaunchPadStaking.sol	da96235149be4a32a1ff620cb11a17c3f4f7483c
PartnersStaking.sol	708072a938f94b84fc449bdc2f0420ce6a550086
xBondlyToken.sol	d3527acd61e67267fc59ace00ef00ed153a23362

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Info	PartnersStaking.sol:135	CS-10	fixed
TBL_SCA_002	Info	PartnersStaking.sol:157	CS-10	fixed
TBL_SCA_003	Info	LaunchPadStaking.sol:42	CS-12	fixed
TBL_SCA_004	Info	BondlyStaking:24	CS-10	fixed

Details

• ID: TBL_SCA-001

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

Function *rewardPerToken()* will be called twice in modifier *updateReward()* when the parameter *account* is not *address(0)*. That would result in the calculation of rewardPerTokenStored being duplicated.

It is recommended to add one more validation in *rewardPerToken()* before the calculation:

```
if (totalStaked == 0 || lastUpdateBlock == block.number) {
        return rewardPerTokenStored;
}
```

• Remediation:

The dev team has updated the contract in the updated version with SHA1 value “124c0c0df89a5fa6daee5107e0976f095d82d3f3”

• ID: TBL_SCA-002

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

Function *_calculateBlocksLeft()* does not need parameters since it is an internal function and only be called by function *_getFutureRewardTokens()* with 2 state variables. These 2 variables can be directly referred to in the Function *_calculateBlocksLeft()* too.

• Remediation:

The dev team has updated the contract in the updated version with SHA1 value “124c0c0df89a5fa6daee5107e0976f095d82d3f3”

• ID: TBL_SCA-003

• Severity: Informational

• Type: CS-12 (Sanity Checks)

• Description:

It is recommended to validate parameter_*amount* to be non-zero in the function *stake()*.

• Remediation:

The dev team has updated the contract in the updated version with SHA1 value “4758304aa000e45081f83e97b6f918280c7f140c”

• ID: TBL_SCA-004

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

The modifier *updateRewardPool()* has duplicated statements. It is recommended to update it as follows:

```
modifier updateRewardPool() {
        if (totalPool > 0) {
            totalPool = totalPool.add(_calculateReward());
        }
        lastUpdateBlock = block.number;
        _;
}
```

• Remediation:

The dev team has updated the contract in the updated version with SHA1 value “7df9ccb7f6809cc362e6594c75021a4136be2961”