Smart Contract Audit Report

for

Bondly.Finance

Version 0.1

Trustlook Blockchain Labs

Email: bd@trustlook.com

Project Overview

Project Name	Bondly.Finance
Contract codebase	N/A
Platform	Ethereum
Language	Solidity
Submission Time	2021.04.29

Report Overview

Report ID	TBL_20210429_00
Version	v0.1
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.04.29
Finished Time	2021.05.02

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. 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)

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 Bonly.Finance

The Bondly.Finance is a data aggregator specially designed for the next generation of payments infrastructure designed for the sale of digital businesses and assets.

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 show in 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.
	CV-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 the issues are described as 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.

Scope

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

File names	Sha1
BondlyAccessControl.sol	34a4e49ba9dec96daa9054de5fe0c838785a8f7a
BondlyStaking.sol	6bc3b0261881607cb8349a0762228f6657afa997
MockBondly.sol	0f7f27efa47bc710fc5a8467ff405a30442d0570
MockNFT1155.sol	d88d7d75fc32f3798b387ea71b22939ed3482cf8
NFTStaking.sol	c050e8e06eeacfb2095b7961f3e684078c85cef4
RewardDistributor.sol	3efa7c07fe275c5589d1699e84fe9a2e77a69034

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	High	BondlyStaking.sol:369	SE-01	unresolved
TBL_SCA_002	High	NFTStaking.sol:318	SE-01	unresolved
TBL_SCA_003	Info	BondlyStaking.sol	CS-10	unresolved

TBL_SCA_004	Info	NFTStaking.sol	CS-10	unresolved
TBL_SCA_005	Info	BondlyStaking.sol:355	AS-05	unresolved
TBL_SCA_006	Info	NFTStaking.sol:304	AS-05	unresolved
TBL_SCA_007	Info	NFTStaking.sol:173	CS-12	unresolved
TBL_SCA_008	Info	BondlyStaking.sol:210	CS-12	unresolved
TBL_SCA_009	Info	NFTStaking.sol:52	CS-08	unresolved
TBL_SCA_010	Info	BondlyStaking.sol:50	CS-08	unresolved

Details

• ID: TBL_SCA-001 - TBL_SCA-002

• Severity: High

• Type: SE-01 (Integer Overflows)

• Description:

In the implementation of rewardOf function, there is a chance of integer underflow. More especially, the issue can be triggered when lastClaimedAt is updated before the timeline of earlyWithdrawal. Since all items in the stake stack will be updated when calling the claimReward function. There is a chance the lastClaimedAt of some item in the stake stack will be updated with a value before the timeline of earlyWithdrawal. Therefore, the following calculation:

```
timePassed = timePassed.sub(
    _stake.lastClaimedAt - earlyWithdrawal - _stake.initTime
);
```

will trigger an integer underflow. Then, the parameter of the sub function will be a huge uint number, which will trigger the Assert() inside the SafeMath library and the rewardOf will fail.

We recommend the rewardOf function checks the value of the lastClaimedAt before operating the subtraction.

• Remediation:

To be discussed

• ID: TBL_SCA-003 - TBL_SCA-004

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

Multiple functions including but not limited to stake(), unstak(), claimRewards() are not called by any of the functions inside the contract.

We recommend using external instead of public for lower gas cost.

• Remediation:

To be discussed

• ID: TBL_SCA-005 - TBL_SCA-006

• Severity: Informational

• Type: AS-05 (Business Logic)

• Description:

In the implementation of rewardOf() function in the NFKStaking.sol and BondlyStaking.sol files. We note that the following validation:

```
if (
    maturityAt <= block.timestamp &&
    _stake.lastClaimedAt >= maturityAt
) continue;
```

This validation will make all stakeholders won't get any rewards after the timeline of fullMaturity (60 days by default) and if the stakeholders' last claim has been done after the timeline of fullMaturity. The only reward stakeholders will get is the time between the earlyWithdrawal and fullMaturity, which is 30 days by default.

We have informed the development team for this finding to make sure the business logic is consistent with the code.

• Remediation:

Nothing changed. The development team has confirmed the logic is consistent with the design.

• ID: TBL_SCA-007 - TBL_SCA-008

• Severity: Informational

• Type: CS-12 (Sanity Checks)

• Description:

For key parameters in the system, it is recommended to add some sanity checks on update. These parameters include but are not limited to minContribution, maxContribution, rewardAPY, mandatoryLock etc.

• Remediation:

To be discussed

• ID: TBL_SCA-009 - TBL_SCA-010

• Severity: Informational

• Type: CS-08 (Event Standard)

• Description:

When defining an Event with address parameters, it is recommended to add indexed key word for them for better query operations.

We advise to update these Events as following:

event Stake(address indexed _staker, uint256 amount);
event Unstake(address indexed _staker, uint256 amount);
event Withdraw(address indexed _staker, uint256 amount);

• Remediation:

To be discussed