Smart Contract Audit Report

for

PETS Token

Version 0.1

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	Bondly.Finance
Contract codebase	N/A
Platform	Ethereum
Language	Solidity
Submission Time	2021.10.05

Report Overview

Report ID	TBL_20211005_00
Version	0.1
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.10.05
Finished Time	2021.10.08

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 PETS Token's smart contracts from Bondly.Finance, 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
PETSToken.sol	31a8a682604263da1dd5acdac3c4cacc1a078f27
PETSTokenLock.sol	cf0b8eb481fb14190c824bccc8f246c58708b066
PETSTokenLock2Rates.sol	e06e6cc012d31efadf88faff5a46d76ffdc60174
PETSTokenLock2Rates2Unlocks.sol	217ede831a873208fe26c297b7bb3b93abb4c5b5
PETSTokenAirdropsLock.sol	eb68ffa19f7675706d98a85e3bffc880699558fb
PETSTokenAngelLock.sol	b84cb62fedf017dfd5d1ef2f2b0b9e2dd12d73f0
PETSTokenFoundationLock.sol	41106a544f3476d6688871c3ed737158283671fa
PETSTokenGamesLock.sol	14b25742e42e785e0a921a3ac984c8a9b5124732
PETSTokenIDOLock.sol	0997c8e94719d449888451987a15dfb88894ad9a
PETSTokenMarketingAndPartnershipsLock.sol	580395310958f814fad239497afaba3cfe94ec56
PETSTokenPrivate1Lock.sol	a27d219fa196e0e813a671cd6db741526355ba75
PETSTokenPrivate2Lock.sol	04bfb92c60042ec06ece14d39260bee3e68ad3d6
PETSTokenRewardsLock.sol	eee72029094ef01317b21fd25f7493d9fdfd36aa
PETSTokenSeedLock.sol	af35e7b2c3dd1205adb1c553c820e992f979699b
PETSTokenStakingLock.sol	2ccab594910fb537ab0c4723eab25fe43d8d714c
PETSTokenTeamAndAdvisorLock.sol	464de0d05e541d32b4459371b4528d249f742050

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	High	PETSToken.sol:23 PETSToken.sol:36	AS-03	fixed
TBL_SCA_002	High	PETSTokenLock.sol:31 PETSTokenLock2Rates.sol:35 PETSTokenLock2Rates2Unlocks.sol:33	AS-03	fixed
TBL_SCA_003	Info	PETSToken.sol:35 PETSToken.sol:44	AS-04	open
TBL_SCA_004	Info	PETSTokenLock.sol:58 PETSTokenLock2Rates.sol:69 PETSTokenLock2Rates2Unlocks.sol:71	CS-08	open

Details

• ID: TBL_SCA-001

• Severity: High

• Type: AS-03 (Semantic Consistency)

• Description:

Function *availableTotalSupply()* and *_availableTotalSupply()* compares the value of tgeAt to block.timestamp. When the value of tgeAt was smaller than block.timestamp, the functions return initialCap. If not, the functions calculate the TotalSupply depends on the passed time after tgeAt. However, tgeAt is bigger than or equal to block.timestamp at this point. Then the later subtraction of “block.timestamp - tgeAt” will result in integer underflow and failure of the calculation.

It is believed the comparison of the 2 values should be as follows::

```
If ( block.timestamp <= tgeAt ){
   return initialCap;
}
```

• Remediation:

The issue has been fixed in commit 7ebb9756717d9721c8b6b48e1e6a32d11efc9b12

• ID: TBL_SCA-002

• Severity: High

• Type: AS-03 (Semantic Consistency)

• Description:

Function *_getAvailableTokens()* in all 3 files compares the value of tgeAt to block.timestamp. When the value of tgeAt was smaller than block.timestamp, the function returns 0. If not, the function calculate the *months* depends on the passed time after tgeAt. However, tgeAt is bigger than or equal to block.timestamp at this point. Then the later subtraction of “block.timestamp - tgeAt” will result in integer underflow and failure of the calculation.

It is believed the comparison of the 2 values should be as follows::

```
If ( block.timestamp <= tgeAt ){
   return 0;
}
```

• Remediation:

The issue has been fixed in commit 7ebb9756717d9721c8b6b48e1e6a32d11efc9b12

• ID: TBL_SCA-003

• Severity: Informational

• Type: AS-04 (Functionality Checks)

• Description:

External functions *availableTotalSupply()* and *availableToMint()* both have identical implementations with their internal functions *_availableTotalSupply()* and *_availableToMint()*.

It is recommended to only keep functions *availableTotalSupply()* and *availableToMint()* and change them to be public.

• Remediation:

• ID: TBL_SCA-004

• Severity: Informational

• Type: CS-08 (Event Standard)

• Description:

Function *send()* did not emit an event for the operation. Recommend emitting events for all the essential state variable updates.

• Remediation: