11th DECEMBER 2020

SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

2 Issues (1 Critical, 0 Major, 1 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io

Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the Neverlose.money smart contract. HAECHI AUDIT conducted the audit focusing on whether Neverlose.money smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.

MAJOR Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.

MINOR Minor issues are some potential risks that require some degree of modification.

TIPS Tips could help improve the code's usability and efficiency

HAECHI AUDIT advises addressing all the issues found in this report.

02. Summary

The code used for the audit can be found at GitHub (https://github.com/Steemhunt/neverlose.money-contract). The last commit for the code audited is at “ea243ff3fe785d7fff801fe5eac510204bf55d02.

Issues

HAECHI AUDIT has 1 Critical Issues, 0 Major Issues, and 1 Minor Issue; also, we included 1 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
CRITICAL	On LockUpPool#addLockUpPool(), WRNRewardPool#addLockUpRewardPool() , it is registrable even if the maxLockUpLimit is 0	(Found - v1.0) (Resolved - v2.0)
MINOR	Using the LockUpPool#addLockUpPool(), new pool can be registered without using WRNRewardPool#addLockUpRewardPool()	(Found - v1.0) (Acknowledged - v2.0)
TIPS	It is recommended to change the contract structure to reduce the code complexity	(Found - v1.0)

Update

[v2.0] - From the new commit a7a9ed1a8dd83ed2b3ece61ada858f7ae171a203, 1 issue has been resolved, and 1 issue has been answered as intended.

03. Overview

Contracts Subject to Audit

- ERC20Token
- LockUpPool
- WRNRewardPool

Key Features

Neverlose.money team has implemented the Smart contract that performs the following functions:

- Pool registration for a specific token
- Deposit tokens in each pool and charge penalties for failure to keep the deposit period
- When the deposit period is observed, distribute the penalty collected in the pool
- Distribute WRN tokens by percentage

Roles

The Neverlose.money Smart contract has the following authorizations:

- **Owner**

The features accessible by each level of authorization is as follows:

Role	MAX	Addable	Deletable	Transferable	Renouncable
Owner	1	X	X	O	O

The functions accessible with the authority are as follows.

Role	Functions
Owner	*LockUpPool#addLockUpPool()* *LockUpPool#updateMaxLimit()* *LockUpPool#setEmergencyMode()* *LockUpPool#setFundAddress()* *WRNRewardPool#addLockUpRewardPool()*

	WRNRewardPool#updatePoolMultiplier() *Ownable#transferOwnership()* *Ownable#renounceOwnership()*

04. Issues Found

CRITICAL: In LockUpPool#addLockUpPool(), WRNRewardPool#addLockUpRewardPool(), maxLockUpLimit can be registered as 0. [Unexpected Behavior] (Found - v1.0)

CRITICAL

Problem Statement

LockUpPool#addLockUpPool(), WRNRewardPool#addLockUpRewardPool() functions take maxLockUpLimit as a factor. At this time, it is checked whether maxLockUpLimit of the pool is 0 in order to confirm that the pool is registered. However, because the function does not fail even if maxLockUpLimit is entered as 0, the same pool can be registered multiple times by entering the value of maxLockUpLimit as 0.

The side effects caused by this are as follows:

- The length of the LockUpPool#pools array becomes longer.
- Because the LockUpPool#pools array becomes longer, the likelihood of the WRNReward#updateAllPools() failure increases.
- In WRNRewardPool#addLockUpRewardPool(), if the multiplier is not 0 and maxLockUpLimit is called multiple times as 0, totalMultiplier and the sum of actual multipliers of each pool will differ.
- In WRNRewardPool#addLockUpRewardPool(), if the multiplier is 255 and maxLockUpLimit is called 258 times to become 0, overflow occurs, decreasing totalMultiplier or even making totalMultiplier become smaller than pool multiplier.

Recommendation

Change the function that registers the pool to fail registration if maxLockUpLimit is 0.

Update

[v2.0] - The issue has been resolved as it has been changed to fail registration if maxLockUpLimit is 0 onLockUpPool#addLockUpPool().

MINOR : By using LockUpPool#addLockUpPool(), new pools can be registered without using WRNRewardPool#addLockUpRewardPool(). (Found - v1.0) (Acknowledged - v2.0)

MINOR

Problem Statement

LockUpPool#addLockUpPool() and WRNRewardPool#addLockUpRewardPool() are both functions for registering the pool. WRNRewardPool#addLockUpRewardPool() function is a function mainly used in operating WRNRewardPool, you can also use LockUpPool#addLockUpPool() function.

Recommendation

It is recommended to change the contract structure and the visibility of functions to ensure consistent use.

Update

[v2.0] - NeverLose.money team is aware of the issue and replied that the above function's visibility is public is intended to conduct unittest.

TIPS : It is recommended to change the contract structure to reduce the code complexity(Found - v1.0)

TIPS

The current implementation has a structure of managing a pool of multiple tokens with one contract. To manage as such, the structures containing information pertinent to the pool are used, and the structures that manage user values per pool are stored separately in each pool. If multiple structures are used in overlap, the contract difficulty increases and readability decreases. To solve this issue, it is recommended to divide between the pool contract that only takes charge of deposit/penalty charge/penalty distribution for each token and the WRNRewrader that distributes WRN to the pool and registers a new pool, so that the pools can be distributed and managed through the WRNRewarder.

05. Disclaimer

This report is not advice on investment, nor does it guarantee the adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.