30th December 2020

SMART CONTRACT AUDIT REPORT

-

version v1.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

0 Issues (0 Critical, 0 Major, 3 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 BadgerDAO smart contract. HAECHI AUDIT conducted the audit focusing on whether BadgerDAO 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/badger-finance/badger-system/). The last commit for the code audited is at "537ef9c44196893cb9760beca2dcc501952e9a4a".

Issues

HAECHI AUDIT has 0 Critical Issues, 0 Major Issues, and 3 Minor Issue; also, we included 3 Tip categories that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
MINOR	**RewardPool#notifyRewardAmount() does not check if it received reward.**	(Found v1.0)
MINOR	**StakingRewards#notifyRewardAmount() can decrease rewardRate**	(Found v1.0)
MINOR	**Numerators can be larger than denominators.**	(Found v1.0)
TIPS	**Controller#revokeStrategy() can be called on active strategy**	(Found v1.0)
TIPS	**Unused Variables**	(Found v1.0)
TIPS	**Remove Test events in StakingRewards**	(Found v1.0)

03. Overview

Contracts Subject to Audit

- BadgerGeyser
- BadgerHunt
- BadgerTree
- BaseStrategy
- ClaimEncoder
- Controller
- EthGifter
- Executor
- HoneypotNft
- MerkleDistributor
- RewardsEscrow
- Sett
- SettAccessControl
- SettAccessControlDefended
- SimpleTimelock
- SingleTokenVestingNonRevocable
- SmartTimelock
- SmartVesting
- StakingRewards
- StrategyBadgerLpMetaFarm
- StrategyBadgerRewards
- StrategyCurveGaugeBase
- StrategyCurveGaugeRenBtcCrv
- StrategyCurveGaugeSbtcCrv
- StrategyCurveGaugeTbtcCrv
- StrategyHarvestMetaFarm
- StrategyPickleMetaFarm
- TokenGifter
- TokenRelease

Roles

The BadgerDAO Smart contract has the following authorizations:

- **Governance**
- **Strategist**
- **AuthorizedActors**
- **AuthorizedPausers**
- **ApprovedStaker**
- **RootUpdater**
- **Guardian**
- **Admin**
- **TokenLocker**

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

Role	Functions
Governance	• Sett#setMin() • Sett#setController() • Controller#approveStrategy() • Controller#revokeStrategy() • Controller#setRewards() • Controller#setSplit() • Controller#setOneSplit() • Controller#setVault() • Controller#setStrategy() • Controller#setConverter() • Controller#withdrawAll() • Controller#inCaseTokensGetStuck() • Controller#inCaseStrategyTokenGetStuck() • StrategyCurveGaugeBase#setKeepCRV() • BaseStrategy#setGuardian() • BaseStrategy#setWithdrawalFee() • BaseStrategy#setPerformanceFeeStrategist() • BaseStrategy#setPerformanceFeeGovernance() • BaseStrategy#setController() • StrategyPickleMetaFarm#setPicklePerformanceFeeGovernance() • StrategyHarvestMetaFarm#setFarmPerformanceFeeGovernance()

	• StrategyHarvestMetaFarm#setFarmPerformanceFeeStrategist() • StrategyPickleMetaFarm#setPicklePerformanceFeeStrategist() • SettAccessControl#setStrategist() • SettAccessControl#setKeeper() • SettAccessControl#setGovernance() • SettAccessControlDefended#approveContractAccess() • SettAccessControlDefended#revokeContractAccess()
Strategist	• Controller#setVault() • Controller#setStrategy() • Controller#setConverter() • Controller#withdrawAll() • Controller#inCaseTokensGetStuck() • Controller#inCaseStrategyTokenGetStuck()
AuthorizedActors	• Sett#earn() • Sett#trackFullPricePerShare() • StrategyCurveGaugeBase#harvest() • StrategyBadgerRewards#harvest() • StrategyBadgerLpMetaFarm#harvest() • BaseStrategy#deposit() • StrategyHarvestMetaFarm#harvest() • StrategyHarvestMetaFarm#tend() • StrategyPickleMetaFarm#harvest() • StrategyPickleMetaFarm#tend()
AuthorizedPausers	• BaseStrategy#pause() • BaseStrategy#unpause()
ApprovedStaker	• StakingRewards#stake()
RootUpdater	• BadgerTree#proposeRoot()
Guardian	• BadgerTree#approveRoot() • BadgerTree#pause() • BadgerTree#unpause()
Admin	• StakingRewards#notifyRewardAmount() • StakingRewards#recoverERC20() • StakingRewards#setRewardsDuration()

	• StakingRewards#pause() • StakingRewards#unpause() • BadgerGeyser#addDistributionToken()
TokenLocker	• BadgerGeyser#signalTokenLock()

04. Issues Found

MINOR : RewardPool#notifyRewardAmount() does not check if it received reward. (Found - v.1.0)

MINOR

```
122.  function notifyRewardAmount(uint256 reward) external updateReward(address(0)) {
123.      _onlyAdmin();
124.      if (block.timestamp >= periodFinish) {
125.          rewardRate = reward.div(rewardsDuration);
126.      } else {
127.          uint256 remaining = periodFinish.sub(block.timestamp);
128.          uint256 leftover = remaining.mul(rewardRate);
129.          rewardRate = reward.add(leftover).div(rewardsDuration);
130.      }
131.
132.      // Ensure the provided reward amount is not more than the balance in the contract.
133.      // This keeps the reward rate in the right range, preventing overflows due to
134.      // very high values of rewardRate in the earned and rewardsPerToken functions;
135.      // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
136.      uint256 balance = rewardsToken.balanceOf(address(this));
137.      emit Test(rewardRate, balance, reward, rewardsDuration);
138.      require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");
139.
140.      lastUpdateTime = block.timestamp;
141.      periodFinish = block.timestamp.add(rewardsDuration);
142.      emit RewardAdded(reward);
143.  }
```

Problem Statement

StakingRewards#notifyRewardAmount() does not check if it has received the reward to distribute. It can lead to a high reward rate for farmers who get rewards faster than others. And can make others unable to earn the rewards.

Since this function is designed to be only called by admin, this error can only be done by admin.

Recommendation

Receive reward token by transferFrom when function is called.

MINOR : StakingRewards#notifyRewardAmount() can decrease rewardRate (Found - v.1.0)

MINOR

```
122.  function notifyRewardAmount(uint256 reward) external updateReward(address(0)) {
123.      _onlyAdmin();
124.      if (block.timestamp >= periodFinish) {
125.          rewardRate = reward.div(rewardsDuration);
126.      } else {
127.          uint256 remaining = periodFinish.sub(block.timestamp);
128.          uint256 leftover = remaining.mul(rewardRate);
129.          rewardRate = reward.add(leftover).div(rewardsDuration);
130.      }
131.
132.      // Ensure the provided reward amount is not more than the balance in the contract.
133.      // This keeps the reward rate in the right range, preventing overflows due to
134.      // very high values of rewardRate in the earned and rewardsPerToken functions;
135.      // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
136.      uint256 balance = rewardsToken.balanceOf(address(this));
137.      emit Test(rewardRate, balance, reward, rewardsDuration);
138.      require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");
139.
140.      lastUpdateTime = block.timestamp;
141.      periodFinish = block.timestamp.add(rewardsDuration);
142.      emit RewardAdded(reward);
143.  }
```

Problem Statement

StakingRewards#notifyRewardAmount() does not check if the rewardRate decreases after notification. Since it updates rate to be *(leftoverRate + notified reward)/duration* when previous reward is not finished, if admin keeps notifying with zero reward, it can lead to continuous decrease on reward rate,

Since this function is designed to be only called by admin, this error can only be done by admin..

Recommendation

Check if rewardRate increases after notifying reward.

MINOR : Numerators can be larger than denominators. (Found - v1.0)

MINOR

Problem Statement

Multiple functions are defined to set numerators used to calculate the percentage of token amount. But these functions do not check if the numerator can be larger than the denominator. Although this issue will only occur when Governance executes functions with bad inputs, it will lead to contracts to revert on normal interactions.

Functions with issue

- Controller#setSplit()
- BaseStrategy#setWithdrawalFee()
- StrategyCurveGaugeBase#initialize()
- StrategyHarvestMetaFarm#initialize()
- StrategyHarvestMetaFarm#setFarmPerformanceFeeGovernance()
- StrategyHarvestMetaFarm#setFarmPerformanceFeeStrategist()
- StrategyPickleMetaFarm#initialize()
- StrategyPickleMetaFarm#setPicklePerformanceFeeGovernance()
- StrategyPickleMetaFarm#setPicklePerformanceFeeStrategist()

Recommendation

Modify function to revert if numerators are larger than denominators.

TIPS: Controller#revokeStrategy() can be called on active strategy (Found - v1.0) TIPS

Controller#revokeStrategy() is used to revoke strategy to prevent old or buggy strategy to be active. Since revoke should mean that the strategy is no longer used, it will be clear to check if the strategy is registered as strategies[_token].

TIPS: Unused Variables (Found - v1.0) TIPS

Controller#{split,max} is defined but not used in contract. Consider removing the variables.

TIPS: Remove Test events in StakingRewards (Found - v1.0) TIPS

StakingRewards contract emits “Test” events which is apparently used for testing. Remove Test events from the contract before deploying.

05. Disclaimer

This report is not an advice on investment, nor does it guarantee 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.