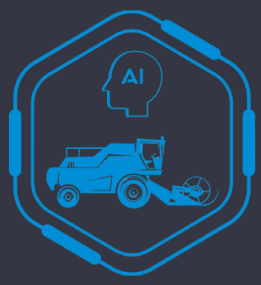
25th NOVEMBER 2020



SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

8 Issues (0 Critical, 1 Major, 7 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 AutoHarvestFinance smart contract. HAECHI AUDIT conducted the audit focusing on whether AutoHarvestFinance 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

Issues

HAECHI AUDIT has 0 Critical Issues, 1 Major Issues, and 7 Minor Issue; also, we included 1 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
MAJOR	CRVStrategyStable#depositArbCheck() always returns true	(Found - v1.0) (Resolved - v2.0)
MINOR	RewardPool#notifyRewardAmount() does not check if it received reward.	(Found - v1.0) (Resolved - v2.0)
MINOR	RewardPool#notifyRewardAmount() can decrease rewardRate	(Found - v1.0)
MINOR	Vault#setVaultFractionToInvest() can not be set to enable full investment.	(Found - v1.0)
MINOR	HardRewards#load() can lead to temporary loss of fund when changing token address	(Found - v1.0)
MINOR	NoMintRewardPool has an Owner which can be misleading against the Governor.	(Found - v1.0)
MINOR	PlayerBook has a Governor role but it is not stored in the Storage contract.	(Found - v1.0)
MINOR	PlayerBook#hasRefer() function name can be declared as misleading.	(Found - v1.0)
TIPS	There are typo errors.	(Found - v1.0)
Notice	Governance role is not Contract and can move user's funds without permission	(Found - v1.0)
Notice	NoMintRewardPool Actually Mints	(Found - v1.0)

03. Overview

Contracts Subject to Audit

Contract File	Sha256 hash
DepositHelper.sol	51ee06b432305fcf8369e4a9e2cc28480e50eca1cd0ceb528c01b67205afc608

PlayerBook.sol	74732fac1da79bc04c636701d75a7573ab2d8a924407e99a5cb2b5f3b657f9d6
RewardToken.sol	c773dff97144cc4be663b633d4e89f664d027af7c6aeef2baef8c2a6353f553f
Controller.sol	d93e41ec3dc6ecd4a85fd2e542e20c6d3fa3ce65542a8842f90722841da949a3
FeeRewardForwarder.sol	c181adda812f0895269256d48a88d392b1cf04433a537547e22d84e4bc7d99d9
Storage.sol	578336dd8b4f442dcb3e4f781af97d98deec80397434d3797a9a39ad79660d07
VaultDAI-stablecoin.sol	e7e2946181abb5981d0c0a2f32a5de7ef88099b7148998a18f23ad91a49e192e
CRVStrategySwerveDAIMainnet.sol	f5a762dfdcc1c4cda3bfaeb52537312019159 53bb11f74d5572d49359f954d65
General_NoMintRewardPool.sol	9f7d46658fed7479ae610bca76c53d54d3a312e1550b7114afb06f8f579a14de

Update

AutoharvestFinance team has provided updated code for CRVStrategySwerveDAIMainnet.sol to fix compile error and 1 Major issue.

Updated hash is
d4405edcde4fc0153576d390c103c9741666a725b0834d1c3af6ecadd48c7b7c

And also provided updated code for General_NoMintRewardPool.sol to fix 1 Minor issue.

Updated hash is

4f09b5b02038d18ec634d64cc866845a2e90c02c1dc30214d2fe6c8c6bb8b572

Roles

The AutoHarvestFinance Smart contract has the following authorizations:

- **Governance**
- **Owner**
- **RewardDistribution**
- **HardWorker**

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

Role	Functions

Governance	• **Storage**
	  ○ setGovernance()
	  ○ setController()
	• **HardRewards**
	  ○ addVault()
	  ○ removeVault()
	  ○ load()
	• **FeeRewardForwarder**
	  ○ setTokenPool()
	• **CRVStrategySwerve**
	  ○ **CRVStrategySwerve**
	  ○ setArbTolerance()
	  ○ salvage()
	  ○ setSell()
	  ○ setSellFloor()
	  ○ createLock()
	  ○ checkPoint()
	  ○ increaseAmount()
	  ○ increaseUnlockTime()
	  ○ withdrawLock()
	• **Controller**
	  ○ addHardWorker()
	  ○ removeHardWorker()
	  ○ addToGreyList()
	  ○ removeFromGreyList()
	  ○ setFeeRewardForwarder()
	  ○ addVaultAndStrategy()
	  ○ doHardWork()
	  ○ rebalance()
	  ○ setHardRewards()
	  ○ salvage()
	  ○ salvageStrategy()
	• **Governable**
	  ○ **Governable**
	  ○ setStorage()
	• **Vault**
	  ○ doHardWork()
	  ○ setStrategy()
	  ○ rebalance()
	  ○ withdrawAll()
	  ○ setVaultFractionToInvest()
	• **RewardToken**
	  ○ addMinter()
	• **PlayerBook**
	  ○ addPool()
	  ○ removePool()
	  ○ setReferRewardRate()
	  ○ setRegistrationStep()

Owner	● **RewardPool** ○ renounceOwnership() ○ transferOwnership() ○ setRewardDistribution()
RewardDistribution	● **RewardPool** ○ notifyRewardAmount()
HardWorker	● **Controller** ○ doHardWork() ○ rebalance()

Notice

- **Governance role is not in audit range, but it is going to be timelock contract with minimum delay of 6 hours**

  Although the Governance role is quite powerful, Autoharvest.finance team is going to migrate the role to a timelock contract which will be deployed in future and

- **NoMintRewardPool Actually Mints**

  NoMintRewardPool can actually mint on notifyRewardAmount and msg.sender is the owner.

04. Issues Found

MAJOR : CRVStrategySwerve#depositArbCheck() always returns true (Found - v1.0) (Resolved - v2.0)

MAJOR

```
function depositArbCheck() public view returns(bool) {
  uint256 currentPrice = wbtcValueFromMixToken(ycrvUnit);
  if (currentPrice > curvePriceCheckpoint) {
    return currentPrice.mul(100).div(wbtcPriceCheckpoint) > 100 - arbTolerance;
  } else {
    return wbtcPriceCheckpoint.mul(100).div(currentPrice) > 100 - arbTolerance;
  }
}
```

Problem Statement

CRVStrategySwerve#depositArbCheck() always returns true as the logic to choose calculation is written in the opposite way.

Recommendation

Change the inequality sign.

Update

Autoharvest.finance team has fixed the issue by applying appropriate sign in new code hash
d4405edcde4fc0153576d390c103c9741666a725b0834d1c3af6ecadd48c7b7c

MINOR : RewardPool#notifyRewardAmount() does not check if it received reward. (Found - v1.0) (Resolved - v2.0)

MINOR

```
function notifyRewardAmount(uint256 reward)
    external
    onlyRewardDistribution
    updateReward(address(0))
{
    if (block.timestamp >= periodFinish) {
        rewardRate = reward.div(duration);
    } else {
        uint256 remaining = periodFinish.sub(block.timestamp);
        uint256 leftover = remaining.mul(rewardRate);
        rewardRate = reward.add(leftover).div(duration);
    }
    lastUpdateTime = block.timestamp;
    periodFinish = block.timestamp.add(duration);

    if(isOwner()) {
        IERC20Mintable(address(rewardToken)).mint(address(this),reward);
        IERC20Mintable(address(rewardToken)).mint(rewardAddr, reward.div(10));
    }
    emit RewardAdded(reward);
}
```

Problem Statement

RewardPool#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 rewardDistribution, this error can only be done by rewardDistribution.

Recommendation

Receive reward token by transferFrom when function is called and rewardDitribution is not owner.

Update

Autoharvest.finance team has fixed the issue by using transferFrom inside the notifyRewardAmount when msg.sender is not owner in updated code hash **4f09b5b02038d18ec634d64cc866845a2e90c02c1dc30214d2fe6c8c6bb8b572**

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

MINOR

```
function notifyRewardAmount(uint256 reward)
    external
    onlyRewardDistribution
    updateReward(address(0))
{
    if (block.timestamp >= periodFinish) {
        rewardRate = reward.div(duration);
    } else {
        uint256 remaining = periodFinish.sub(block.timestamp);
        uint256 leftover = remaining.mul(rewardRate);
        rewardRate = reward.add(leftover).div(duration);
    }
    lastUpdateTime = block.timestamp;
    periodFinish = block.timestamp.add(duration);
    if(isOwner()) {
        IERC20Mintable(address(rewardToken)).mint(address(this),reward);
        IERC20Mintable(address(rewardToken)).mint(rewardAddr, reward.div(10));
    }
    emit RewardAdded(reward);
}
```

Problem Statement

RewardPool#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 rewardDistribution keeps notifying with zero reward, it can lead to continuous decrease on reward rate,

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

Recommendation

Check if rewardRate increases after notifying reward.

MINOR : Vault#setVaultFractionToInvest() can not be set to enable full investment. (Found - v.1.0)

MINOR

```
 function setVaultFractionToInvest(uint256 numerator, uint256 denominator) external
onlyGovernance {

   require(denominator > 0, "denominator must be greater than 0");

   require(numerator < denominator, "denominator must be greater than numerator");

   vaultFractionToInvestNumerator = numerator;

   vaultFractionToInvestDenominator = denominator;

 }
```

Problem Statement

Because of the second require statement, vault can not invest the full amount of deposits

Recommendation

Change require statement to include when numerator is same as denominator

MINOR : HardRewards#load() can lead to temporary loss of fund when changing token address (Found - v.1.0)

MINOR

```
function load(address _token, uint256 _rate, uint256 _amount) external onlyGovernance {
  token = IERC20(_token);
  blockReward = _rate;
  if (address(token) != address(0) && _amount > 0) {
    token.safeTransferFrom(msg.sender, address(this), _amount);
  }
}
```

Problem Statement

HardRewards#load() changes the token address that will be used to reward the hard workers. But, when the token address is changed, it does not give back the original token which can be resolved by changing back to original token, but this could lead to malfunction if other hardworker is rewarded between changes.

Recommendation

Transfer original token back to controller or governance when token address has changed.

MINOR : NoMintRewardPool has an Owner which can be misleading against the Governor. (Found - v.1.0)

MINOR

Problem Statement

NoMintRewardPool inherits Controllable and Owable which makes it has both Owner and Governance.
Since other contracts can be controlled by Controllable which checks storage that acts as a registry to track the governance address.
By adding Owner on RewardPool can lead to complexity in operation

Recommendation

Do not inherit the Ownable on NoMintRewardPool

MINOR : PlayerBook has a Governor role but it is not stored in the Storage contract. (Found - v.1.0)

MINOR

Problem Statement

PlayerBook has a Governor role but it is not guaranteed to match the Governor role stored in the Storage contract.

Recommendation

Inherit Controllable contract and use Governor of the Storage contract for consistency.

MINOR : PlayerBook#hasRefer() function name can be declared as misleading. (Found - v.1.0)

MINOR

```
function hasRefer(address from)
    isRegisteredPool()
    external
    returns(bool)
{
    _determinePID(from);
    uint256 pID =  _pIDxAddr[from];
    return (_plyr[pID].laff > 0);
}
```

Problem Statement

Current implementation of PlayerBook#hasRefer() checks if *from* has a refer but If there is no *pID* corresponding to *from*, it allocates from to the latest pid, which makes the function cannot be restricted to view.

Since function name implies the function is view function, which does not change the storage value, process of allocating the new pid can be misleading

Recommendation

Remove the *_determinePID(from)* and restrict the function to view.

TIPS : There are typos (Found - v.1.0)

TIPS

```
function getPlayerName(address from) external view returns (bytes32)
{
    uint256 pID =  _pIDxAddr[from];
    lif(_pID==0){
        return "";
    }
    return (_plyr[pID].name);
}

function getPlayerLaffAddress(address from) external view returns(address laffAddress) {
    uint256 pID =  _pIDxAddr[from];
    lif(_pID==0){
        return _teamWallet;
    }
    uint256 laffID = _plyr[pID].laff;
    if(laffID == 0) {
        return _teamWallet;
    }
    return _plyr[laffID].addr;
}

function getPlayerLaffName(address from) external view returns (bytes32)
{
    uint256 pID =  _pIDxAddr[from];
    lif(_pID==0){
        return "";
    }
    uint256 aID=_plyr[pID].laff;
    if( aID== 0){
        return "";
    }
    return (_plyr[aID].name);
}

function getPlayerInfo(address from) external view returns (uint256,uint256,uint256)
```

Problem Statement

These four functions are implemented to return a specified value when the *pID* of the input is 0 since 0 *pID* means the user does not have a *pID*. However, each function is using the global variable *_pID* which indicates the latest pid issued for users, instead of the *pID* which came in as a parameter.

Recommendation

Change *_pID* into *pID*.

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.