14th JANUARY 2021

SMART CONTRACT AUDIT REPORT

-

version v2.1

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

5 Issues (2 Critical, 1 Major, 2 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 armor smart contract. HAECHI AUDIT conducted the audit focusing on whether armor 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 resolving all the issues found in this report.

02. Summary

Scope of audit

The code used for the audit was provided as archive files.

arNXM

hash : fe9dcc2652332e3389584a159aff75f711f281acecd45b802b1a213da8a36b32

armor-core/arShield

hash : b23824a73cb52627733ad5c91d4520c76f72860c3472c50fcc98ea160f795233

Update

[v2.0] - Armor team has provided updated version of arNXM(RefferalRewards.sol, TimelockOwned.sol, arNXMVault.sol, Ownable.sol) and fixed 2 issues and confirmed that 1 TIPS is intended bahavior.

[v2.1] - Armor team has provided updated version of Armor-Core and fixed 2 issues

Issues

HAECHI AUDIT has 2 Critical Issues, 1 Major Issues, and 2 Minor Issue; also, we included 1 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
CRITICAL	ExpireTracker/BalanceExpireTracker does not work properly	(Found - v1.0) (Fixed - v2.1)
CRITICAL	arNXM can fail to transfer if user has received arNXM without deposit	(Found - v1.0) (Fixed - v2.0)
MAJOR	BalanceManager#keep(), StakeManager#keep() will process only 3 expiry	(Found - v1.0) (Intended - v2.1)
MINOR	TimelockOwned cannot call function that needs ether	(Found - v1.0) (Fixed - v2.0)

MINOR	SafeMath will not be applied properly on uint64, uint128	(Found - v1.0)
TIPS	RewardReferral#withdraw(),stake() does not process the referral parameter	(Found - v1.0) (Intended - v2.0)

03. Overview

Features

- ArNXM
    - stake wNXM
    - stake nxm to PooledStaking based on unwrapped wNXM
    - ability to withdraw PooledStaking without waiting to full withdraw period
- Armor-Core
    - stake/withdraw arNFT
    - subscribe to insurance by paying ether
    - claim coverage when hack happend
- ArShield
    - stake Uniswap lp token and get covered by unwrapping lp token and paying the underlying tokens.

Roles

The armor Smart contract has the following authorizations:

- **Owner**

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

Role	Functions

Owner	• arShield
	  ○ withdrawBalance()
	• RewardManagerWithReferral
	  ○ setRewardDistribution()
	  ○ changeReferPercent()
	  ○ rescueToken()
	• TimelockOwned
	  ○ implementProposal()
	  ○ submitProposal()
	• PlanManager
	  ○ adjustMarkup()
	• StakeManager
	  ○ allowProtocol()
	  ○ changeWithdrawalDelay()
	  ○ toggleUF()
	• ClaimManager
	  ○ confirmHack()
	• BalanceManager
	  ○ changeRefPercent()
	  ○ changeGovPercent()
	  ○ changeDevPercent()
	  ○ toggleUF()
	  ○ toggleShield()
	• ArmorMaster
	  ○ registerModule()
	  ○ addJob()
	  ○ deleteJob()
	• ArmorModule
	  ○ changeMaster()
	• GovernanceStaker
	  ○ setRewardDistribution()
	• UtilizationFarm
	  ○ setRewardDistribution()
	• LPFarm
	  ○ setRewardDistribution()

04. Issues Found

CRITICAL : ExpireTracker/BalanceExpireTracker does not work properly (Found - v1.0)(Fixed - v2.1)

CRITICAL

```
59.     function push(uint160 expireId, uint64 expiresAt)
60.     internal
61.   {
62.     require(expireId != EMPTY, "info id address(0) cannot be supported");
63.
64.     // If this is a replacement for a current balance, remove it's current link first.
65.     if (infos[expireId].expiresAt > 0) _remove(expireId);
66.     ...
153.       while(checkPoints[prevCursor].tail != EMPTY){
154.         prevCursor = uint64( prevCursor.sub(BUCKET_STEP) );
155.       }
```

[BalanceExpireTracker.sol]

```
73.     if (infos[head].expiresAt >= expiresAt) {
74.       // pushing nft is going to expire first
75.       // update head
76.       infos[expireId] = ExpireMetadata(0,head,expiresAt);
77.       head = expireId;
78.
79.       // update head of bucket
80.       Bucket storage b = checkPoints[bucket];
81.       b.head = expireId;
82.
83.       if(b.tail == 0) {
84.         // if tail is zero, this bucket was empty should fill tail with expireId
85.         b.tail = expireId;
86.       }
87.
88.       // this case can end now
89.       return;
90.     }
91.
92.     // then check if depositing nft will last more than latest
93.     if (infos[tail].expiresAt <= expiresAt) {
94.       // push nft at tail
95.       infos[expireId] = ExpireMetadata(tail,0,expiresAt);
96.       tail = expireId;
97.
98.       // update tail of bucket
99.       Bucket storage b = checkPoints[bucket];
100.      b.tail = expireId;
101.
102.      if(b.head == 0){
103.        // if head is zero, this bucket was empty should fill head with expireId
104.        b.head = expireId;
105.      }
106.
107.      // this case is done now
108.      return;
```

```
109.      }
110.      …
142.        while(checkPoints[prevCursor].tail != 0){
143.          prevCursor = uint64( prevCursor.sub(BUCKET_STEP) );
144.        }
```

[ExpireTracker.sol]

Problem Statement

We found some bugs in ExpireTracker/BalanceExpireTracker

1. BalanceExpireTracker.sol#65 - use pop(expireId) instead of _remove(expireId) using _remove will lead to many unintended behavior like empty bucket but non-zero tail/head
2. BalanceExpireTracker.sol#153 - change "!=" to "=="
3. ExpireTracker.sol#76 - change "ExpireMetadata(0,head,expiresAt)" to "ExpireMetadata(head,0,expiresAt)" and add "infos[head].prev = expireId"
4. ExpireTracker.sol#95 - change "ExpireMetadata(tail,0,expiresAt)" to "ExpireMetadata(0,tail,expiresAt)" and add "infos[tail].next = expireId"
5. ExpireTracker.sol#143 - change "!=" to "=="

Update

[v2.1] - Armor team has fixed the above unintended behaviors

CRITICAL : arNXM can fail to transfer if user has received arNXM without deposit (Found - v1.0)(Fixed - v2.0)

CRITICAL

```
280.     function alertTransfer(address _from, address _to, uint256 _amount)
281.    external
282.  {
283.     require(msg.sender == address(arNxm), "Sender must be the token contract.");
284.
285.     // address(0) protection is for mints and burns.
286.     if ( referrers[_from] != address(0) ) rewardManager.withdraw(referrers[_from], _from, _amount);
287.     if ( referrers[_to] != address(0) ) rewardManager.stake(referrers[_to], _to, _amount);
288.  }
```

Problem Statement

We have found some circumstance where user's transfer will fail

let's assume that user A has referrer and B does not have referrer at the beginning

1. user A send user B 100 arNXM
    a. since user B does not have referrer, rewardManager.stake() is not called
2. user B deposits 1 wNXM and receives 1 arNXM and registers user C as referrer
    a. user B now has referrer
3. user B tries to transfer 100 arNXM back to user A
    a. since user C does not have stake in rewardManager, tx fails

This will result in a locked asset in user B that makes B unable to transfer arNXM received before deposits.

Update

[v2.0] - Armor team has fixed this issue by changing ReferralRewards#withdraw() function to change the withdrawing value to be staked amount when input value is larger than staked amount

MAJOR : BalanceManager#keep(), StakeManager#keep() will process only 3 expiry (Found - v1.0)(Intended - v2.1) MAJOR

Problem Statement

keep() functions only process 3 expiry, if the actual expired amount was more than 3, it will result in inconsistent values. Although the limit of process seems to be the result of gas optimization, it should be noted since this can lead to unintended behavior

Recommendation

We recommend finding a way to process all expiry while maintaining reasonable gas usage.

Update

[v2.1] - Armor team has confirmed this is intended behavior to save gas when user interacts with the contract

MINOR : TimelockOwned cannot call function that needs ether (Found - v1.0) (Fixed - v2.0) MINOR

Problem Statement

TimelockOwned#implementProposal() executes function that was registered before. But submitProposal does not store the “value” which can be used while executing the proposal.

Recommendation

get “value” as parameter on TimelockOwned#submitProposal() and make executeProposal() call the function with sending “value” amount of ether.

Update

[v2.0] - Armor team has fixed this issue by adding “value” parameter in submitProposal()

MINOR : SafeMath will not be applied properly on uint64, uint128 (Found - v1.0) MINOR

Problem Statement

SafeMath library is only applicable to uint256, but we found some contracts using SafeMath for uint64/uint128.

Recommendation

Implement SafeMath for uint64/uint128 and use that library instead

TIPS: RewardReferral#withdraw(),stake() does not process the referral parameter (v2.0 - Intended)

TIPS

Problem Statement

RewardReferral#withdraw(),stake() gets “referral” as parameter but does not process any storage related actions besides the events. Although overall functionalities are not affected by this, but it should be noted for the clarity if this is intended behavior.

Update

[v2.0] - Armor team has confirmed that this behavior is intended which is to just log the referral in the event

05. Disclaimer

This report is not 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.