HAECHI AUDIT

NINANCE.IO

Smart Contract Security Analysis

Published on : Jan 12, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

NINANCE.IO

Security Report Published by HAECHI AUDIT
v2.0 Jan 12, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	3	3	-	-	All Issues are Resolved
Minor	3	3	-	-	All Issues are Resolved
Tips	3	-	-	-	-

TABLE OF CONTENTS

6 Issues (0 Critical, 3 Major, 3 Minor) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit@haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by Ninance team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Ninance team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends the Universe team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/NinanceService/Audit_code_base/tree/7b1b724963f5ac899d61b6af64e8c16c5d84f751). The last commit of the code used for this Audit is "7b1b724963f5ac899d61b6af64e8c16c5d84f751".

Issues	HAECHI AUDIT found 0 critical issue, 3 major issues, and 3 minor issues. There are 3 tips issues explained that would improve the code's usability or efficiency upon modification.
Update	[v.2.0] In commit "d4e6c33513552c550792fb6cfd37f774346d493" of the new github repository(https://github.com/NinanceService/contract), major issues, and 3 minor issues have been revised.

Severity	Issue	Status
⚠ MAJOR	ERANFT#officalMint() function causes compilation error.	(Found - v1.0) (Resolved - v2.0)
⚠ MAJOR	The REAUpdateCard#getUpdateRecord(), NFTExchange#getTakerOrder() functions fail to operate properly.	(Found - v1.0) (Resolved - v2.0)
⚠ MAJOR	An error may occur if Grade is 0 in NFT_MasterChef#_stake().	(Found - v1.0) (Resolved - v2.0)
● MINOR	The NFT_MasterChef#BONUS_MULTIPLIER variable is unintentionally fixed to 1 and cannot be changed.	(Found - v1.0) (Resolved - v2.0)

MINOR	In the ERANFT#officalMint() function, the NFT sales proceeds are moved to the Promote address, possibly causing the ERANFT#distribute() function called internally in that function fail to operate properly.	(Found - v1.0) (Resolved - v2.0)
MINOR	The AuctionEnded event of the NFTExchange#auctionEnd() function returns an invalid event parameter.	(Found - v1.0) (Resolved - v2.0)
TIPS	Function names contain typographical errors.	(Found - v1.0)
TIPS	Variable names contain typographical errors.	(Found - v1.0)
TIPS	require() statements contain typographical errors.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- ERC20
- IERC20
- ERC721
- IERC165
- IERC721
- IERC721Metadata
- IERC721TokenReceiver
- IERC721TokenReceiverEx
- Address
- Context
- SafeERC20
- SafeMath
- String
- Util
- ContractOwner
- ERANFT
- ERAToken
- Manager
- Member
- LimitTimeAuction
- NFTExchange
- Promote
- Blacklist
- Ownable
- Pausable
- PauserRole
- Roles
- ERC20Pausable
- Token
- REAUpdateCard

FINDINGS

⚠ MAJOR

ERANFT#officalMint() function causes compilation error. (Found - v.1.0) (Resolved - v.2.0)

```
181   function officalMint(string memory _hash) public returns(uint256){              //
官方创建
182       require(isSold[_hash] == false, "Sold");
183       require(isInitialized == true, "Init contarct first");
184       require(paused == false, "offical mint is paused");
185       require(block.timestamp >= officalStart,"NOT start!");
186       address user = msg.sender;
187       uint256 NFTprice = 525*1e18;
188       uint256 era_price = getPrice();
189       uint256 needPay = NFTprice.mul(1e18).div(era_price);
190
IERC20(era).transferFrom(user,address(manager.members("PromoteAddress")),needPay);
191       distribute(needPay);
192       mintinternal(user,user,hash,1,0,50,true);
193       emit OfficalMint(user,user, hash, 1, NFT_Id, needPay);
194       return NFT_Id;
195   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/ERAnft.sol#L181-L195]

Issue

The hash inside *ERAnft.sol:192-193* is a variable that does not exist, causing a compilation error.

Recommendation

We recommend changing the hash in *ERAnft.sol:192-193* to _hash.

Update

The variable name has been changed normally, so compilation errors no longer occur.

⚠ MAJOR

The REAUpdateCard#getUpdateRecord(), NFTExchange#getTakerOrder() functions fail to operate properly. (Found - v.1.0) (Resolved - v.2.0)

```
187   function getUpdateRecord(uint256 tokenId) public view returns(CardUpdateRecord[]
memory){
188       (,,uint256 Grade,,,) = nft.starAttributes(tokenId);
189       require(Grade ⩾ 2,"No upgrade record!");
190       CardUpdateRecord[] memory record;
191       for(uint256 i = 2;i⩽Grade;i++){
192           record[i] = updateRecord[tokenId][i];
193       }
194       return record;
195
196   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/UpdateCard.sol#L187-L196]

```
   function getTakerOrder(uint256 tokenid) public view returns(TakerOrder[] memory) {
       TakerOrder[] memory takerorders;
       for(uint256 i=0;i<auctionOrdersArray[tokenid].length;i++){
               address taker = auctionOrdersArray[tokenid][i];
               TakerOrder memory order = takerOrders[tokenid][taker];
               takerorders[i] = order;

       }
       return takerorders;
   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/NftMarket.sol#L371-L380]

Issue

The *REAUpdateCard#getUpdateRecord()*, *NFTExchange#getTakerOrder()* functions receive the necessary information inside the contract as an array. In declaring the array to be returned inside the functions, it ends up having a default value. The default value of the array is an empty array because the declaration is made without specifying the length. In both functions, an invalid opcode error occurs because the access is attempted to an empty array using the index operator.

Recommendation

It is recommended to reallocate the array as the required length after accessing via the index and before assigning a value.

Update

The two functions are deleted, so there is no longer an issue.

⚠ MAJOR

An error may occur if Grade is 0 in NFT_MasterChef#_stake(). (Found - v.1.0) (Resolved - v.2.0)

```
857   function _stake(uint256 _pid,uint256[] memory amount) internal returns(uint256 totalAmount){
858       PoolInfo storage pool = poolInfo[_pid];
859       UserInfo storage user = userInfo[_pid][msg.sender];
860       uint256 len = amount.length;
861       uint256 tokenID;
862       require(len <= maxWithAmount,"can not big then maxWithAmount");
863       for(uint256 i=0;i<len;i++){
864           tokenID = amount[i];
865           require(!isInStating[msg.sender][tokenID],"already exit");
866           require(INFT(pool.lpToken).ownerOf(tokenID) == msg.sender,"not ownerOf");
867           lockedTime[tokenID] = block.timestamp;
868           (,,uint256 Grade,,,) = INFT(pool.lpToken).starAttributes(tokenID);
869           totalWeight += nftweight[Grade - 1];
870
871           totalAmount = totalAmount.add(nftweight[Grade - 1]);
872
873           isInStating[msg.sender][tokenID] = true;
874           user._nftBalances.push(tokenID);
875           emit DepositNFT(msg.sender,_pid,tokenID);
876       }
877
878       pool.totalStakingNFTAmount = pool.totalStakingNFTAmount.add(len);           //更新总抵押nft数量
879       pool.lpSupply = pool.lpSupply.add(totalAmount);
880       INFT(pool.lpToken).batchTransferFrom(msg.sender,address(this),amount);
881   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/nftStakingReward.sol#L857-L881]

Issue

The *NFT_MasterChef#_stake()* function is called when the *NFT_MasterChef#deposit()* function is called. This function deposits NFTs in the staking pool and updates the deposit information.

It brings in the grade of the NFT to be deposited in the NFT contract from *nftStakingReward.sol:868*. Afterward, using the grade in *nftStakingReward.sol:871*, it determines how much weight the NFT has.

However, when Grade is 0, an integer overflow occurs, thereby causing abnormal index access. Thus, function execution may not be performed normally.

Recommendation

We recommend adding a statement that checks cases where the Grade is not greater than 1.

Update

A require statement has been added to check whether Grade is 0, so there is no longer an issue.

MINOR

The NFT_MasterChef#BONUS_MULTIPLIER variable is unintentionally fixed to 1 and cannot be changed. (Found - v.1.0) (Resolved - v.2.0)

```
762   function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
763       if (_to ⩽ bonusEndBlock) {
764           return _to.sub(_from).mul(BONUS_MULTIPLIER);
765       } else if (_from ⩾ bonusEndBlock) {
766           return _to.sub(_from);
767       } else {
768           return bonusEndBlock.sub(_from).mul(BONUS_MULTIPLIER).add(
769               _to.sub(bonusEndBlock)
770           );
771       }
772   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/nftStakingReward.sol#L762-L772]

Issue

The *NFT_MasterChef#BONUS_MULTIPLIER* variable is used in the *NFT_MasterChef#getMultiplier()* function, which is a function that enables confirmation of the reward for depositing for a specific period.
BONUS_MULTIPLIER is multiplied by the existing reward up to the bonus block specified when the NFT_MasterChef contract is deployed for the first time.
However, because BONUS_MULTIPLIER is set to 1 and this value cannot be changed, BONUS_MULTIPLIER exists as a meaningless variable.

Recommendation

If BONUS_MULTIPLIER will not be used, we recommend deleting the logic related to the variable. If BONUS_MULTIPLIER will be used, we recommend adding a function that can change the value.

Update

The BONUS_MULTIPLIER variable has been deleted, so the issue no longer occurs.
However, since the *NFT_MasterChef#getReward()* function returns the value of _to.sub(_from) in any branch, we recommend removing the branch statement for optimization.

MINOR

In the ERANFT#officalMint() function, the NFT sales proceeds are moved to the Promote address, possibly causing the ERANFT#distribute() function called internally in that function to fail to operate properly.

(Found - v.1.0)

```
181   function officalMint(string memory _hash) public returns(uint256){              //
官方创建
182       require(isSold[_hash] == false, "Sold");
183       require(isInitialized == true, "Init contarct first");
184       require(paused == false, "offical mint is paused");
185       require(block.timestamp >= officalStart,"NOT start!");
186       address user = msg.sender;
187       uint256 NFTprice = 525*1e18;
188       uint256 era_price = getPrice();
189       uint256 needPay = NFTprice.mul(1e18).div(era_price);
190
IERC20(era).transferFrom(user,address(manager.members("PromoteAddress")),needPay);
191       distribute(needPay);
192       mintinternal(user,user,hash,1,0,50,true);
193       emit OfficalMint(user,user, hash, 1, NFT_Id, needPay);
194       return NFT_Id;
195   }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/ERAnft.sol#L181-L195]

Issue

The *ERANFT#officalMint()* allows users to purchase NFTs in a specific period. When a user purchases NFT through this function, the information is recorded in the ERANFT contract, and the sales proceeds received from the user are delivered to the address set as PromoteAddress. Afterward, the ERANFT#distribute() function is called to send a part of the sales proceeds to the addresses set as OfficalAddress and PromoteAddress. Because the sales proceeds are sent to the address set by PromoteAddress, not to the ERANFT contract, the function will always fail if the ERANFT contract has no USDT.

Recommendation

Similar to the *ERANFT#preOfficialMint()* function, we recommend modifying the sales proceeds to be sent to the ERANFT contract.

Acknowledgement

If the intention was to send the sales proceeds through the *ERANFT#officalMint()* function to the address set as PromoteAddress, no modification is necessary.

Update

Now that tokens are delivered normally to the ERANFT contract, there is no longer an issue.

MINOR

The AuctionEnded event of the NFTExchange#auctionEnd() function returns an invalid event parameter. (Found - v.1.0) (Resolved - v.2.0)

```
297   function auctionEnd(uint256 tokenid,address taker) public payable{
298
299       require(msg.sender == makerOrders[tokenid].maker, "No permission to end");
300       require(msg.sender == tx.origin,"Only EOA!");
301       uint8 payment = makerOrders[tokenid].payment;
302       if(auctionOrdersArray[tokenid].length >0){
303           IERC721(manager.members("nft")).transferFrom(address(this),taker,tokenid);
304
305           uint256 tradeFee =
takerOrders[tokenid][taker].auctionAmount.mul(txFees).div(1000);    //收取2.5%的手续费
306
307           (address origin,,,,uint256 stampFees,) =
INFT(manager.members("nft")).starAttributes(tokenid);
308
309           uint256 stampFee =
takerOrders[tokenid][taker].auctionAmount.mul(stampFees).div(1000);
310           uint256 sendAmount =
takerOrders[tokenid][taker].auctionAmount.sub(tradeFee).sub(stampFee);
311
312           if (payment == 0) {
313               payable(manager.members("funder")).transfer(tradeFee);
314               payable(origin).transfer(stampFee);
315               payable(makerOrders[tokenid].maker).transfer(sendAmount);
316
317               for(uint256 i=0;i<auctionOrdersArray[tokenid].length;i++){
//退还其他拍卖出价
318
319                   if(auctionOrdersArray[tokenid][i] == taker){
320                       delete takerOrders[tokenid][auctionOrdersArray[tokenid][i]];
321                       continue;                                                    //结束此
次循环到下次循环
322                   }
323                   else{
324                       address otherTaker = auctionOrdersArray[tokenid][i];
325                       uint256 backAmount =
takerOrders[tokenid][otherTaker].auctionAmount;
326                       payable(otherTaker).transfer(backAmount);
327                       delete takerOrders[tokenid][otherTaker];
328                   }
329
330               }
```

```
331             } else {
332                 IERC20 token;
333                 if (payment == 1) {
334                     token = usdt;
335                 }
336                 if (payment == 2) {
337                     token = era;
338                 }
339                 IERC20(token).transfer(manager.members("funder"), tradeFee);
340                 IERC20(token).transfer(origin, stampFee);
341                 IERC20(token).transfer(makerOrders[tokenid].maker,sendAmount);
342 
343                 for(uint256 i=0;i<auctionOrdersArray[tokenid].length;i++){
//退还其他拍卖出价
344 
345                     if(auctionOrdersArray[tokenid][i] == taker){
346                        delete takerOrders[tokenid][auctionOrdersArray[tokenid][i]];
347                         continue;                                                    //结束此
次循环到下次循环
348                     } else{
349                         address otherTaker = auctionOrdersArray[tokenid][i];
350                         uint256 backAmount =
takerOrders[tokenid][otherTaker].auctionAmount;
351                         IERC20(token).transfer(otherTaker, backAmount);
352                         delete takerOrders[tokenid][otherTaker];
353                     }
354 
355                 }
356             }
357 
358         }
359         else{       //流拍
360 
IERC721(manager.members("nft")).transferFrom(address(this),makerOrders[tokenid].maker,to
kenid);
361         }
362 
363         emit AuctionEnded(taker,takerOrders[tokenid][taker].auctionAmount, tokenid,
payment);
364         delete makerOrders[tokenid];
365         delete auctionOrdersArray[tokenid];
```

```
366     }
```

[https://github.com/NinanceService/Audit_code_base/blob/7b1b724963f5ac899d61b6af64e8c16c5d84f751/NftMarket.sol#L297-L366]

Issue

The *NFTExchange#auctionEnd()* function selects an address that has made a favorable offer among the price offers for NFTs posted by the seller, receives the amount offered by the address, and sells NFTs.

When the function ends normally, the AuctionEnded event occurs. And, as a second parameter, takerOrders[tokenId][taker].auctionAmount is returned. This means the amount offered by the taker. However, because the information of takerOrders[tokenId][taker] is deleted before the corresponding event occurs, the value is always returned as 0.

Recommendation

We recommend either to make an event occur before deleting *takerOrders[tokenId][taker]* information or to make an event occur by saving the auctionAmount of the struct in another variable.

Update

Before deleting the *takerOrder[tokenId][taker]* information, the AuctionEnded event occurred normally and no longer issues occurred.

💡 **TIPS**

Function names contain typographical errors. (Found - v.1.0)

The *ERANFT#officalMint()* function is confirmed to be a typo of the ERANFT#officialNFT() function.
The *ERANFT#pauseOfficalMint()* function is confirmed to be a typo of the ERANFT#pauseOfficialMint() function.

💡 **TIPS**

Variable names contain typographical errors. (Found - v.1.0)

The *ERANFT#PreOlder_Price* variable is confirmed to be a typo of ERANFT#PreOrder_Price.
The *ERANFT#officalStart* variable is confirmed to be a typo of ERANFT#officialStart.
The _officalStart variable, the second parameter of the *ERCNFT#init()* function, is confirmed to be a typo of the _officialStart variable.

💡 **TIPS**

require() statements contain typographical errors. (Found - v.1.0)

The "pre must earlier than offical!" explanation of the require() statement of *ERAnft.sol:105* is confirmed to be a typo of "pre must earlier than official!".
The "Init contarct first" explanation of the require() statement of *ERAnft.sol:170* is confirmed to be a typo of "Init contract first".
The "Init contarct first" explanation of the ()require statement of *ERAnft.sol:183* is confirmed to be a typo of "Init contract first".
The "offical mint is paused" explanation of the ()require statement of *ERAnft.sol:184* is confirmed to be a typo of "official mint is paused".
The "offical NFT can only support ERA token!" explanation of the ()require statement of *NftMarket.sol:106* is confirmed to be a typo of "official NFT can only support ERA token".
The "NO Enlough Time to lock '' explanation of the ()require statement of *nftStakingReward.sol:995* is confirmed to be a typo of "NO Enough Time to lock".
The "can not bing self" explanation of the ()require statement of *promote.sol:378* is confirmed to be a typo of "can not bind self".

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the Mainnet. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
  Address
    #isContract()
      ✓ returns false for account address
      ✓ returns true for contract address
    #sendValue()
      ✓ should fail if try to transfer more than sender contract amounts
      when sender contract has ethers
        ✓ sends 0 wei
        ✓ sends non-zero amounts
        ✓ sends the whole balance
        ✓ should fail if try to send more than the amounts
      with contract recipient
        ✓ sends ether (41ms)
        ✓ shoudl fail if recipient reverts (38ms)
  Ownable
    #constructor()
      ✓ should set msg.sender to owner
    #transferOwnership()
      ✓ should fail when msg.sender is not owner
      ✓ should fail if try to transfer ownership to ZERO_ADDRESS
      ✓ should change owner to newOwner
      ✓ should emit transferOwnership event
SafeMath
    add
      ✓ adds correctly
      ✓ reverts on addition overflow
    sub
      ✓ subtracts correctly
      ✓ reverts if subtraction result would be negative
    mul
      ✓ multiplies correctly
      ✓ multiplies by zero correctly
      ✓ reverts on multiplication overflow
    div
      ✓ divides correctly
```

```
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on division by zero
  mod
    ✓ reverts with a 0 divisor
    modulos correctly
      ✓ when the dividend is smaller than the divisor
      ✓ when the dividend is equal to the divisor
      ✓ when the dividend is larger than the divisor
      ✓ when the dividend is a multiple of the divisor
SafeERC20
  if address is not contract
    ✓ should fail on transfer
    ✓ should fail on transferFrom
    ✓ should fail on approve
  with token that returns false on all calls
    ✓ should fail on transfer
    ✓ should fail on transferFrom
    ✓ should fail on approve
  with token that returns true on all calls
    ✓ should not fail on transfer
    ✓ should not fail on transferFrom
  with token that returns no boolean values
    ✓ should not fail on transfer
    ✓ should not fail on transferFrom
 Address
  #functionCall()
    with valid contract receiver
      ✓ calls the requested function
      ✓ reverts when the called function reverts with no reason
      ✓ reverts when the called function reverts, bubbling up the revert reason
      ✓ reverts when the called function runs out of gas
      ✓ reverts when the called function throws
      ✓ reverts when function does not exist
    with non-contract receiver
      ✓ reverts when address is not a contract
  #functionCallWithValue()
    with zero value
      ✓ calls the requested function
    with non-zero value
      ✓ reverts if insufficient sender balance
      ✓ calls the requested function with existing value (42ms)
      ✓ calls the requested function with transaction funds
      ✓ reverts when calling non-payable functions (39ms)
```

```
ERANFT
  #init()
    ✓ should fail if msg.sender is not owner (40ms)
    ✓ should fail if already initialized (51ms)
    ✓ should fail if preStart >= officialStart
  #preOfficialMint()
    ✓ should fail if already sold (4636ms)
    ✓ should fail if contract is not initialized
    ✓ should fail if presale is not start (47ms)
    ✓ should fail if sale is ended
    valid case
      ✓ usdt move to ERCNFT (1204ms)
      ✓ should emit PreMint event (1093ms)
  #officalMint()
    ✓ should fail if already sold (2210ms)
    ✓ should fail if contract is not initialized
    ✓ should fail if presale is not start (47ms)
    ✓ should fail if official mint is paused (70ms)
    ✓ should fail if sale is ended
    valid case
      ✓ nft move to user (1990ms)
      ✓ should emit OfficalMint event (963ms)
  #userMint()
    ✓ should fail if stamp fee out of range
    valid case
      ✓ nft move to user
      ✓ should emit UserMint event (38ms)

LimitTimeAuction
  #createAuction()
    ✓ should fail if auction already exist (91ms)
    ✓ should fail if auction end time is already passed
    ✓ should fail if auction start time > auction end time
    ✓ should fail if payment is not ERA token when user try to create auction with official nft
    ✓ should fail if user does not approve nft
    ✓ should fail if non nft owner try to create auction
    valid case
      ✓ nft moves to auction contract (54ms)
      ✓ should emit CreateOrder (50ms)
  #bid()
    ✓ should fail if auction not started
    ✓ should fail if auction already ended
    ✓ should fail if try to bid lower than highest bid amount
    ✓ should fail if bidder does not approve payment (1899ms)
```

```
    valid case
      ✓ bid information update (1584ms)
      ✓ before highest bidder's bid amount refunded (838ms)
      ✓ bidder's amount transfer to auction contract (1086ms)
      ✓ should emit HighestBidIncreased (946ms)
  #cancelAuction()
    ✓ should fail if auction ended
    ✓ should fail if msg.sender is not auction maker
    valid case
      ✓ if bidder exist, refund (49ms)
      ✓ nft return to owner (50ms)
      ✓ should emit CancelAuction (47ms)
  #auctionEnd()
    ✓ should fail if auction not ended
    valid case
      ✓ if bidder exist, nft transfer to highest bidder (82ms)
      ✓ nft origin get stamp fee (79ms)
      ✓ funder get trade fee (67ms)
      ✓ auction maker get bid amount except fee (68ms)
  #setTxFee()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ change tx fee
      ✓ should emit ChangeTxFee event

NFT_MasterChef
  #add()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if lpToken is zero address
    valid case
      ✓ pool length updated
      ✓ pool information updated
  #set()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to set non-existent pool
    valid case
      ✓ pool information update
  #changeRewardFee()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to set reward fee more than 20 percent
    valid case
      ✓ reward fee changed
      ✓ should emit ChangeFee event
  #changeFloateFee()
```

```
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to set reward fee more than 20 percent
   valid case
     ✓ floate fee changed
  #deposit()
    ✓ should fail if length of tokenID list is zero
    ✓ should fail If you try to deposit a token that is not owned by you (116ms)
   valid case
     ✓ nft move to chef contract
     ✓ pool information update
  #withdraw()
    ✓ should fail if msg.sender is not token owner
    ✓ should fail if not enough time passed (145ms)
   valid case
     ✓ nft move to user
     ✓ user reward debt information update
  deposit/withdraw/claim integration test
    ✓ user information update properly
    ✓ user claim token

NFTExchange
  #createOrder()
    ✓ should fail order already exist (105ms)
    ✓ should fail if payment is not ERA token when user try to create order with offical nft
    ✓ should fail if user does not approve nft (39ms)
    ✓ should fail if tradePrice is not max value of uint256 when canExchange flag is false
   valid case
     ✓ nft moves to exchange contract (60ms)
     ✓ should emit CreateOrder event (59ms)
  #changePrice()
    ✓ should fail if try to change price of non existent tokenid
    ✓ should fail if msg.sender is not order creater
    ✓ should fail if canExchange flag is false
   valid case
     ✓ price changed
  #takeOrder()
    ✓ should fail if try to take non existent order
    ✓ should fail if pay amount mismatch (42ms)
   valid case
     ✓ nft transfer to taker (72ms)
     ✓ nft origin get stamp fee (73ms)
     ✓ funder get trade fee (74ms)
     ✓ auction maker get bid amount except fee (74ms)
     ✓ maker order information deleted (69ms)
```

```
    ✓ should emit TradeOrder event (110ms)
  #cancelOrder()
    ✓ should fail if msg.sender is not order maker
    valid case
      ✓ order information deleted
      ✓ should emit CancelOrder
  #setTxFee()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if new tx fee is too high
    valid case
      ✓ change tx fee
      ✓ should emit ChangeTxFee event
  #bid()
    valid case
      ✓ bid amount transfer to auction contract
      ✓ bid information update
      ✓ should emit HighestBidIncreased event
  #cancelBid()
    ✓ should fail if no bid information
    valid case
      ✓ bid information update
      ✓ bid amount refund
  #auctionEnd()
    ✓ should fail if non auction maker try to end the auction
    valid case
      ✓ taker get nft
      ✓ non taker bid amount refund
      1) should emit AuctionEnded event

Promote
  #bind()
    ✓ should fail if binding address is not in the game
    ✓ should fail if try to binding msg.sender
    ✓ should fail if already binded (47ms)
    valid case
      ✓ f/gs/ss information update properly
      ✓ should emit NewJoin event
  #redem()
    ✓ should fail if redem address is not in the game
    ✓ should fail if msg.sender is not pool
    valid case
      ✓ possibly user level change
  #newDeposit()
    ✓ should fail if contract call from pool
```

```
    ✓ should fail if origin is not in game
   valid case
     ✓ user.f, user.ff rewards claimed
     ✓ should emit NewValid event
  #deposit()
    ✓ should fail if amount is zero
    ✓ should fail if user does not approve promote contract (1039ms)
    ✓ should fail msg.sender does not gamer
   valid case
     ✓ daliy information update
     ✓ should emit Deposit event
  #withdraw()
    ✓ should fail if withdraw locked
   valid case
     ✓ withdraw era
     ✓ should emit Withdraw event

 REAUpdateCard
  #mintCard()
    ✓ should fail if wrong grade
    ✓ should fail if user does not approve card contract (1142ms)
   valid case
     ✓ user get card
     ✓ should emit Mintcard event
  #updateNFT()
    ✓ should fail if nft is not official nft
    ✓ should fail if try to update nft that is not owned by user
    ✓ should fail if grade mismatch
    ✓ should fail if card is not mine
   valid case
     ✓ card burn
     ✓ power update
     ✓ should emit Update
  #getUpdateRecord
    2) index error
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/ERC20					
ERC20.sol	100	100	100	100	

IERC20.sol	100	100	100	100	
contracts/ERC721					
ERC721.sol	100	100	100	100	
IERC165.sol	100	100	100	100	
IERC721.sol	100	100	100	100	
IERC721Metadata.sol	100	100	100	100	
IERC721TokenReceiver.sol	100	100	100	100	
IERC721TokenReceiverEx.sol	100	100	100	100	
contracts/lib					
Address.sol	100	100	100	100	
Context.sol	100	100	100	100	
SafeERC20.sol	100	100	100	100	
SafeMath.sol	100	100	100	100	
String.sol	100	100	100	100	
Util.sol	100	100	100	100	
contracts/					
ContractOwner.sol	100	100	100	100	
ERAToken.sol	100	100	100	100	
ERAnft.sol	100	100	100	100	
Manager.sol	100	100	100	100	
Member.sol	100	100	100	100	
NftAuction.sol	100	100	100	100	
NftMarket.sol	100	100	100	100	
UpdateCard.sol	100	100	100	100	

nftStakingReward.sol	100	100	100	100	
promote.sol	100	100	100	100	
token.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document