HAECHI AUDIT

SHOYU

Smart Contract Security Analysis

Published on : Sep 6, 2021

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

SHOYU

Security Report Published by HAECHI AUDIT

v1.0 Aug 23, 2021

v2.0 Sep 6, 2021

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	2	2	-	-	All issues resolved
Minor	3	3	-	-	All issues resolved
Tips	-	-	-	-	-

TABLE OF CONTENTS

5 Issues (0 Critical, 2 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 Sushiswap,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 Shoyu smart contract created by SushiSwap team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by SushiSwap 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 SushiSwap 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/sushiswap/shoyu). The last commit of the code used for this Audit is c7804e7c2626e778cab96b6a4811fddacc7f41aa.

Issues

HAECHI AUDIT found 0 critical issues, 2 major issues, and 3 minor issues. There are 0 Tips issues explained that would improve the code's usability or efficiency upon modification.

Update

[v.2.0] Regarding new commit
9f3e161eae80dbb413bb6ef371cac35c590d4da1,
fc9c2671d4b621e2707f9bedcb8d80f2a688d268,
36ec672a520f75a75c72291383213af71d85bf7569f,
fixed 2 major issues and 3 minor issues.

Severity	Issue	Status
⚠ **MAJOR**	Due to Strategy logic, the auction may not work properly.	(Found - v1.0) (Resolved - v2.0)
⚠ **MAJOR**	When the factory address differs from the owner address, TokenFactory contract cannot distribute NFT721 and NFT1155.	(Found - v1.0) (Resolved - v2.0)
○ **MINOR**	Due to the absence of *IERC721#onERC721Received()* function implementation, the auction is not performed normally.	(Found - v1.0) (Resolved - v2.0)
○ **MINOR**	Due to the absence of *IERC1271#isValidSignature()* function implementation, the auction is not performed normally.	(Found - v1.0) (Resolved - v2.0)
○ **MINOR**	Anyone can issue NFTs using Factory Contract.	(Found - v1.0) (Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- ERC20SnapshotInitializable
- ReentrancyGuardInitializable
- ProxyFactory
- ERC20Initializable
- BaseNFT721
- ERC721GovernanceToken
- NFT721
- TokenHelper
- NFT1155
- ITokenFactory
- IBaseNFT721
- IBaseNFT1155
- BaseNFT1155
- INFT1155
- INFT721
- PaymentSplitter
- BaseExchange
- TokenFactory
- ERC721Liquidator
- IERC721GovernanceToken
- DividendPayingERC20.sol
- IDividendPayingERC20
- ERC721Initializable
- ISocialToken
- SocialToken
- IPaymentSplitter
- ERC1155Exchange
- Orders
- IBaseExchange
- ERC721Exchange

Shoyu smart contract has the following privileges.

- Owner

Each privilege can access the following functions.

Role	Functions
Owner	Ownable#renounceOwnership() Ownable#transferOwnership() NFT1155#setRoyaltyFeeRecipient() NFT1155#setRoyaltyFee() NFT721#setRoyaltyFeeRecipient() NFT721#setRoyaltyFee() TokenFactory#setBaseURI721() TokenFactory#setBaseURI1155() TokenFactory#setProtocolFeeRecipient() TokenFactory#setOPerationalFeeRecipient() TokenFactory#setStrategyWhitelisted() SocialToken#mint() BaseNFT1155#setBaseURI() BaseNFT721#setBaseURI()

FINDINGS

⚠ MAJOR

Due to Strategy logic, auction may not work properly.

(Found - v.1.0) (Resolved - v.2.0)

```
function _bid(
        Orders.Ask memory askOrder,
        bytes32 askHash,
        address bidder,
        uint256 bidAmount,
        uint256 bidPrice,
        address bidRecipient,
        address bidReferrer
    ) internal returns (bool executed) {
        require(canTrade(askOrder.token), "SHOYU: INVALID_EXCHANGE");
        require(bidAmount > 0, "SHOYU: INVALID_AMOUNT");
        require(amountFilled[askHash] + bidAmount <= askOrder.amount, "SHOYU: SOLD_OUT");

        _validate(askOrder, askHash);
        _verify(askHash, askOrder.signer, askOrder.v, askOrder.r, askOrder.s);

        if (IStrategy(askOrder.strategy).canExecute(askOrder.deadline, askOrder.params, bidder,
bidPrice)) {
            amountFilled[askHash] += bidAmount;

            address recipient = askOrder.recipient;
            if (recipient == address(0)) recipient = askOrder.signer;
            require(
                _transferFeesAndFunds(askOrder.currency, bidder, recipient, bidPrice * bidAmount),
                "SHOYU: FAILED_TO_TRANSFER_FUNDS"
            );

            if (bidRecipient == address(0)) bidRecipient = bidder;
            _transfer(askOrder.token, askOrder.signer, bidRecipient, askOrder.tokenId, bidAmount);

            emit Execute(askHash, bidder, bidAmount, bidPrice, bidRecipient, bidReferrer);
            return true;
        }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/base/BaseExchange.sol#L120]

```
function canExecute(
        uint256 deadline,
        bytes memory,
        address,
        uint256
    ) external view override returns (bool) {
        return deadline < block.number;
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/strategies/EnglishAuction.sol#L8-L15]

Issue

When the NFT auction begins by Governance Token, users who intend to participate in the bid can do so through the *BaseExchange#bid()* function. The currently implemented *BaseExchange#bid()* function works as follows:

1. Verifies that the bid intending to participate is valid. (_verify, _validate)
2. Through the *canExcute()* function of the registered strategy contract, it checks a successful bidder of the concerned auction.
3. When the *canExecute()* function returns true, the NFT is received and the bid price is paid.
4. When the *canExecute()* function returns false, it checks whether the offered bid price is higher than the existing highest bid. When the bid price is updated, it records the information in the contract.

In sum, the end of the auction is determined according to the implementation of the *canExecute()* function. The EnglishAuction contract, one of the Strategies implemented as of now, returns true when the auction deadline has elapsed.

In this case, regardless of the bid price, the NFT will be awarded to the user who bids first after the deadline. That is, a situation outside a typical auction scenario occurs.

Recommendation

Please appropriately modify the *Strategy* logic.

Update

[v2.0] - According to new commits 9f3e161eae80dbb413bb6ef371cac35c590d4da1 and fc9c2671d4b621e2707f9bedcb8d80f2a688d268, the *canExcute()* function was changed to the *canClaim()* function. Logic for comparison is added to the *canClaim()* function, therefore, this problem is resolved.

⚠ MAJOR

When the factory address differs from the owner address, TokenFactory contract cannot distribute NFT721 and NFT1155.

(Found - v.1.0) (Resolved - v.2.0)

```
function initialize(
        string memory _name,
        string memory _symbol,
        address _owner,
        uint256 toTokenId,
        address royaltyFeeRecipient,
        uint8 royaltyFee
    ) external override initializer {
        __BaseNFTExchange_init();
        initialize(_name, _symbol, _owner);

        _parkTokenIds(toTokenId);

        emit ParkTokenIds(toTokenId);

        setRoyaltyFeeRecipient(royaltyFeeRecipient);
        setRoyaltyFee(royaltyFee);
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/NFT721.sol#L33-L50]

```
 function setRoyaltyFeeRecipient(address royaltyFeeRecipient) public override onlyOwner {
        require(royaltyFeeRecipient != address(0), "SHOYU: INVALID_FEE_RECIPIENT");

        _royaltyFeeRecipient = royaltyFeeRecipient;

        emit SetRoyaltyFeeRecipient(royaltyFeeRecipient);
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/NFT721.sol#L78-L84]

```
function setRoyaltyFee(uint8 royaltyFee) public override onlyOwner {
        if (_royaltyFee == type(uint8).max) {
            require(royaltyFee <= ITokenFactory(_factory).MAX_ROYALTY_FEE(), "SHOYU: INVALID_FEE");
        } else {
            require(royaltyFee < _royaltyFee, "SHOYU: INVALID_FEE");
        }

        _royaltyFee = royaltyFee;

        emit SetRoyaltyFee(royaltyFee);
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/NFT721.sol#L86-L96]

Issue

TokenFactory Contract is a contract to distribute and manage multiple NFT721, NFT1155 Contracts. The basic initialize process is as follows:

1. Call *TokenFactory#createNFT721()* or *TokenFactory#createNFT1155()* from TokenFactory.
2. Initialize NFT transaction-related codes, initialize NFT metadata, and set contract owner in the NFT contract initializer
3. Perform the early mining process
4. Set the account that receives royalty fees and the fee ratio

Step 4 above includes *onlyOwner()* modifier and therefore can be executed only by the contract owner.

However, when the TokenFactory address differs from the owner contract of the NFT contract, msg.sender becomes the address of the TokenFactory while the NFT contract owner becomes the address of the owner received as a parameter during the NFT initialize process, causing Step 4 not to be executed. Thus, the initialization process fails to be performed normally.

Recommendation

It is recommended either to allow the code execution of the factory in *setRoyaltyFeeRecipient()*, *setRoyaltyFee()* or to call by categorizing the concerned functions into public functions and internal functions.

Update

[v2.0] - In the new commit 36ec672a520f75a75c72291383213af71d85bf9f , the problem is solved by separating the setRoyaltyFeeRecipient and setRoyaltyFee functions into a public functions that can only be called by the owner and an internal function that is called from the constructor.

MINOR

Due to the absence of IERC721#onERC721Received() function implementation, the auction is not performed normally.

(Found - v.1.0) (Resolved - v.2.0)

```
function liquidate(
        address nft,
        uint256 tokenId,
        uint8 minimumQuorum
    ) external override returns (address proxy) {
        bytes memory initData =
            abi.encodeWithSignature(
                "initialize(address,address,address,uint256,uint8)",
                factory,
                orderBook,
                nft,
                tokenId,
                minimumQuorum
            );
        proxy = _createProxy(_target, initData);

        IERC721(nft).safeTransferFrom(msg.sender, proxy, tokenId);

        emit Liquidate(proxy, nft, tokenId, minimumQuorum);
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/ERC721Liquidator.sol#L41]

Issue

ERC721Liquidator contract creates Governance Token containing NFTs to be submitted for auction.

When you call by putting the address of the NFT contract address to submit for the auction and Token ID into *ERC721Liquidator#Liquidate()* as parameters, a Governance Token contract corresponding to the NFT is created and NFT is sent to the contract.

The *IERC721#safeTransferFrom()* function is used in this process. This function checks whether the *IERC721#onERC721Received()* function of the contract returns a normal value when the token receiver is a non-EOA contract address.

However, the *onERC721Received()* function is currently not implemented in ERC721GovernanceToken, NFT transmissions fail. That is, Governance Token cannot be created using the ERC721Liguidator contract.

Recommendation

Please implement the *ERC721GovernanceToken#onERC721Received()* function.

Update

[v2.0] - With the code update, the ERC721Liquidator contract is deprecated.

MINOR

Due to the absence of IERC1271#isValidSignature() function implementation, the auction is not performed normally.

(Found - v.1.0) (Resolved - v.2.0)

```
function _verify(
        bytes32 hash,
        address signer,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal view {
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR(), hash));
        if (Address.isContract(signer)) {
            require(
                IERC1271(signer).isValidSignature(digest, abi.encodePacked(r, s, v)) == 0x1626ba7e,
                "SHOYU: UNAUTHORIZED"
            );
        } else {
            require(ecrecover(digest, v, r, s) == signer, "SHOYU: UNAUTHORIZED");
        }
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/base/BaseExchange.sol#L214]

Issue

As explained above, when a user intending to auction bids by using the *BaseExchange#bid()* function, it verifies the validity of the bid. In the *BaseExchange#_verify()* function, it verifies the validity of the signature of ask. When the signer of ask is contract, it verifies with the return value of the *IERC1271#isValidSignature()* function.

According to the current structure of Shoyu contract, it is highly likely that the signer of ask becomes Governance Token. However, because there are no implements of the isvalidSignature() function in Governance Token, the *BaseExchange#_verify()* function always fails, causing the auction not to be performed normally.

Recommendation

Please implement the *ERC721GovernanceToken#isValidSignature()* function.

Update

[v2.0] - ERC721Liquidator contract is deprecated due to code update.

MINOR

Anyone can issue NFTs using Factory Contract.

(Found - v.1.0) (Resolved - v.2.0)

```
function mintWithTags721(
        address nft,
        address to,
        uint256 tokenId,
        bytes memory data,
        string[] memory tags
    ) external override {
        _setTags(nft, tokenId, tags);
        IBaseNFT721(nft).mint(to, tokenId, data);
    }
```

[https://github.com/sushiswap/shoyu/blob/c7804e7c2626e778cab96b6a4811fddacc7f41aa/contracts/TokenFactory.sol#L223]

Issue

For NFT tokens distributed by TokenFactory, only the owner of TokenFactory and NFT contract can execute *NFT721#mint()* and *NFT1155#mint()*. However, because anyone can call the *TokenFactory#mintWithTags721()*, *TokenFactory#mintWithTags1155()* functions, anyone can issue as many NFTs as they want.

Recommendation

When calling *TokenFactory#mintWithTags721()* or *TokenFactory#mintWithTags1155()*, it is advised to check whether the call is from an address allowed to mint via modifier or signature.

Acknowledgement

If this is intended behavior, no further modification is necessary.

Update

[v2.0] -In a new commit 2f22f1c05643684a50787436e756ba4efbc02357, a new role called deployer was added. The problem has been solved by changing to only address registered as a deployer to issue NFTs.

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 in Binance Smart Chain and Solidity. To write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover 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 having issues.

```
DividendPayingERC20
  #mint()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if recipient is ZERO_ADDRESS
    valid case
      ✓ totalSupply should increase
      ✓ recipient balance should increase
      ✓ accumulative dividend does not update
      ✓ withdrawable dividend does not update
  #burn()
    ✓ should fail if burner does not have enough value
    valid case
      ✓ totalSupply should decrease
      ✓ recipient balance should decrease
      ✓ accumulative dividend does not update
      ✓ withdrawable dividend does not update
  #transfer()
    ✓ should fail if recipient is ZERO_ADDRESS
    ✓ should fail if sender does not have enough balance
    valid case
      ✓ sender balance should decrease
      ✓ recipient balance should increase

ERC20Initializable
  #constructor()
    ✓ should set name properly
    ✓ should set symbol properly
    ✓ should set decimals properly
    ✓ should set initial supply properly
  ERC20 Spec
    #transfer()
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than balance
      when succeeded
        ✓ sender's balance should decrease
        ✓ recipient's balance should increase
        ✓ should emit Transfer event
    #transferFrom()
      ✓ should fail if sender is ZERO_ADDRESS
```

```
        ✓ should fail if recipient is ZERO_ADDRESS
        ✓ should fail if sender's amount is lower than transfer amount
        ✓ should fail if allowance is lower than transfer amount
        ✓ should fail even if try to transfer sender's token without approval process
        when succeeded
          ✓ sender's balance should decrease
          ✓ recipient's balance should increase
          ✓ should emit Transfer event
          ✓ allowance should decrease
          ✓ should emit Approval event
      #approve()
        ✓ should fail if spender is ZERO_ADDRESS
        valid case
          ✓ allowance should set appropriately
          ✓ should emit Approval event
      #increaseAllowance()
        ✓ should fail if spender is ZERO_ADDRESS
        ✓ should fail if overflows
        valid case
          ✓ allowance should set appropriately
          ✓ should emit Approval event
      #decreaseAllowance()
        ✓ should fail if spender is ZERO_ADDRESS
        ✓ should fail if overflows
        valid case
          ✓ allowance should set appropriately
          ✓ should emit Approval event

  governance
    #submitSellProposal()
      ✓ should fail if expired
      ✓ should fail if insufficient power
      valid case
        ✓ proposal submitted
        ✓ totalPower of proposal id update
        ✓ power of proposer with respect to proposal id update
        ✓ should emit SubmitSellProposal event
    #confirmSellProposal()
executed
      ✓ should fail if already executed
      ✓ should fail if expired
      ✓ should fail if not submitted proposal
      ✓ should fail if user already confirmed
      ✓ should fail if insufficient power
      valid case
        ✓ totalPower of proposal id update
        ✓ power of proposer with respect to proposal id update
        ✓ should emit SubmitSellProposal event
        ✓ executed if total power exceeds minPower
  #revokeSellProposal()
   ✓ should fail if already executed
   ✓ should fail if not confirmed user try to revoke
   valid case
    ✓ totalPower of proposal id update
    ✓ power of proposer with respect to proposal id update
    ✓ should emit RevokeSellProposal event
  #execute()
   ✓ should fail if already executed
   ✓ should fail if expired
   ✓ should fail if not enough power
   valid case
    ✓ should emit ExecuteSellProposal event
  Bidding Process
   bidding with arguments
    ✓ should fail if cannot bid (50ms)
    when canExecute
     1)   should fail if non-highest bidder try to payout
     ✓ payout (44ms)

 NFT721
  #constructor()
   ✓ register ERC1155 interface
  #balanceOf()
   ✓ should fail if query for ZERO_ADDRESS
   ✓ returns 0 if user does not have any tokens
   ✓ returns the amount of tokens owned by user
  #balanceOfBatch()
   ✓ should fail if query for ZERO_ADDRESS
   ✓ should fali if length mismatch
   ✓ returns 0 if user does not have any tokens
   ✓ returns the amount of tokens owned by user
  #setApprovedForAll()
   ✓ should fail if try to approve msg.sender
   valid case
    when approved flag is true
     ✓ approved for all
     ✓ approved user can transfer any tokens
     ✓ should emit ApprovalForAll event
    when approved flag is false
     ✓ approved for all clear
     ✓ should emit ApprovalForAll event
  #safeTransferFrom()
   ✓ should fail if msg.sender is not owner nor approved
   ✓ should fail if recipient is ZERO_ADDRESS
   ✓ should fail if CA recipient has invalid return value (43ms)
   ✓ should fail if CA recipient reverts (38ms)
   when recipient: EOA
    normal transferFrom executed
    ✓ token balance change
    ✓ should emit TransferSingle event
   when recipient: CA with _ERC1155_RECEIVED return value
    normal transferFrom executed
     ✓ token balance should change
     ✓ should emit TransferSingle event
 #safeBatchTransferFrom()
  ✓ should fail if msg.sender is not owner nor approved
  ✓ should fail if recipient is ZERO_ADDRESS
  ✓ should fail if CA recipient has invalid return value (43ms)
  ✓ should fail if CA recipient reverts (39ms)
  when recipient: EOA
   normal transferFrom executed
    ✓ token balance change
    ✓ should emit TransferBatch event
   when recipient: CA with _ERC1155_BATCH_RECEIVED return value
    normal transferFrom executed
     ✓ token balance should change
     ✓ should emit TransferBatch event
 #mint()
  ✓ should fail if msg.sender is not owner
  valid case
   ✓ token mint
 #mintBatch()
  ✓ should fail if msg.sender is not owner
  ✓ should fail if mint fail
  valid case
   ✓ token mint
 #burn()
  ✓ should fail if msg.sender does not have token with respect to tokenId
  ✓ should fail if msg.sender does not have enough amount token to burn
  valid case
   ✓ token burn
  #burnBatch()
   ✓ should fail if msg.sender does not have token with respect to tokenId
   ✓ should fail if msg.sender does not have enough amount token to burn
   valid case
    ✓ token burn
  #permit()
   valid case
    ✓ approved

NFT721
 #constructor()
  ✓ register ERC721 interface
 #balanceOf()
  ✓ should fail if query for ZERO_ADDRESS
  ✓ returns 0 if user does not have any tokens
  ✓ returns the amount of tokens owned by user
```

```
#ownerOf()
  ✓ should return ZERO_ADDRESS if query for nonexistent token
  ✓ returns token owner
#approve()
  ✓ should fail if try to approve token owner
  ✓ should fail if msg.sender is not owner nor approved for all
  valid case
    ✓ approved
    ✓ approved user can transfer token
    ✓ should emit Approval event
#setApprovedForAll()
  ✓ should fail if try to approve msg.sender
  valid case
    when approved flag is true
      ✓ approved for all
      ✓ approved user can transfer any tokens
      ✓ should emit ApprovalForAll event
    when approved flag is false
      ✓ approved for all clear
      ✓ should emit ApprovalForAll event
  #transferFrom()
    ✓ should fail if msg.sender is not owner nor approved
    ✓ should fail if recipient is ZERO_ADDRESS
    valid case
      ✓ token owner should change
      ✓ sender's token balance should decrease
      ✓ recipient's token balance should decrease
      ✓ token approval should clear
      ✓ should emit Transfer event
  #safeTransferFrom()
    ✓ should fail if msg.sender is not owner nor approved
    ✓ should fail if recipient is ZERO_ADDRESS
    ✓ should fail if CA recipient has invalid return value (44ms)
    ✓ should fail if CA recipient reverts (39ms)
    when recipient: EOA
      normal transferFrom executed
        ✓ token owner should change
        ✓ sender's token balance should decrease
        ✓ recipient's token balance should decrease
        ✓ token approval should clear
        ✓ should emit Transfer event
      when recipient: CA with _ERC721_RECEIVED return value
        normal transferFrom executed
          ✓ token owner should change
          ✓ sender's token balance should decrease
          ✓ recipient's token balance should decrease
          ✓ token approval should clear
          ✓ should emit Transfer event
#mint()
```

```
  ✓ should fail if msg.sender is not owner
  ✓ should fail if try to mint already minted token
  valid case
   ✓ token mint
 #mintBatch()
  ✓ should fail if msg.sender is not owner
  ✓ should fail if mint fail
  valid case
   ✓ token mint
 #burn()
  ✓ should fail if msg.sender does not have token with respect to tokenId
  valid case
   ✓ token burn
 #burnBatch()
  ✓ should fail if msg.sender does not have token with respect to tokenId
  valid case
   ✓ token burn
 #permit()
  valid case
   ✓ approved
 #permitAll()
  valid case
   ✓ all tokens approved

Ownable
 #constructor()
  ✓ should set msg.sender to owner
 #transferOwnership()
  ✓ should fail if msg.sender is not owner
  ✓ should fail if try to transfer ownership to AddressZero
  valid case
   ✓ should change owner to newOwner
   ✓ should emit OwnershipTransferred event
 #renounceOwnership()
  ✓ should fail if msg.sender is not owner
  valid case
   ✓ should change owner to AddressZero
   ✓ should emit OwnershipTransferred event

PaymentSplitter
 #release()
  ✓ should fail if contract does not have enough tokens
  valid case
   ✓ account get tokens
   ✓ should emit PaymentReleased event

TokenFactory
 #createNFT721()
  initialize with minting multiple nfts
```

```
      valid case
        ✓ clone NFT721 (42ms)
        ✓ nft721 owner set
        ✓ nft721 royaltyFee information set
        ✓ nft mints
    initialize with parking nfts
      valid case
        ✓ clone NFT721
        ✓ nft721 owner set
        ✓ nft721 royaltyFee information set
        ✓ nft parked (68ms)
  #createNFT1155()
    valid case
      ✓ clone NFT1155 (40ms)
      ✓ nft1155 owner set
      ✓ nft1155 royaltyFee information set
      ✓ nft mints
  #createSocialToken()
    valid case
      ✓ clone SocialToken
      ✓ socialToken owner set
      ✓ dividendToken address set
  #mintWithTags721()
    2) should fail if msg.sender is not factory nor nft contract owner
    valid case
      ✓ nft mints
      ✓ set tags
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/base					
BaseExchange.sol	100	100	100	100	
BaseNFT721.sol	100	100	100	100	
BaseNFT1155.sol	100	100	100	100	
DividendPayingERC20.sol	100	100	100	100	
ERC20Initializable.sol	100	100	100	100	
ERC721Initializable.sol	100	100	100	100	
ERC1155Initializable.sol	100	100	100	100	
OwnableInitializable.sol	100	100	100	100	

ProxyFactory.sol	100	100	100	100	
ReentrancyGuardInitializable.sol	100	100	100	100	
contract/interfaces					
IBaseExchange.sol	100	100	100	100	
IBaseNFT721.sol	100	100	100	100	
IBaseNFT1155.sol	100	100	100	100	
IDividendPayingERC20.sol	100	100	100	100	
IERC721GovernanceToken.sol	100	100	100	100	
IERC721Liquidator.sol	100	100	100	100	
IERC1271.sol	100	100	100	100	
INFT721.sol	100	100	100	100	
INFT1155.sol	100	100	100	100	
IOrderBook.sol	100	100	100	100	
IPaymentSplitter.sol	100	100	100	100	
ISocialToken.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
ITokenFactory.sol	100	100	100	100	
contract/libraries					
Orders.sol	100	100	100	100	
TokenHelper.sol	100	100	100	100	
contract/strategies					
DesignatedSale.sol	100	100	100	100	
DutchAuction.sol	100	100	100	100	
EnglishAuction.sol	100	100	100	100	

FixedPriceSale.sol	100	100	100	100	
contract/					
ERC721Exchange.sol	100	100	100	100	
ERC721GovernanceToken.sol	100	100	100	100	
ERC721Liquidator.sol	100	100	100	100	
ERC1155Exchange.sol	100	100	100	100	
NFT721.sol	100	100	100	100	
NFT1155.sol	100	100	100	100	
OrderBook.sol	100	100	100	100	
PaymentSplitter.sol	100	100	100	100	
SocialToken.sol	100	100	100	100	
TokenFactory.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document