Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.03.20, the SlowMist security team received the Flooring Lab team's security audit application for Bitmap Punks, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

BitmapPunks is an NFT project on the Ethereum blockchain that combines innovative mechanisms of non - fungible tokens (NFTs) and fungible tokens. Each BitmapPunk NFT corresponds to one BMP token, and 1:1 BMP tokens are automatically airdropped upon minting.

Transferring or selling BMP tokens will transfer the corresponding BitmapPunk NFT into the burn pool. The decoupled tokens can randomly retrieve an image from the burn pool at any time through the "reveal" function.

Transferring or selling a BitmapPunk NFT will automatically transfer the corresponding BMP tokens and lock the NFT.

The feature and attribute data of the NFT are generated by on-chain encoding.

Users can make a offer to sell the locked NFTs they hold, and they can also bid for the purchase of other users' NFTs.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Lack of fee settlement when minting NFTs	Design Logic Audit	Critical	Fixed
N2	Missing update numExchangableNFT when minting NFTs	Design Logic Audit	High	Fixed
N3	Missing chainID check in the signature verification	Replay Vulnerability	Medium	Fixed
N4	Incorrect condition for numExchangableNFT update in the _exchangeNFT function	Design Logic Audit	High	Fixed
N5	Pseudo-random risk	Block data Dependence Vulnerability	Medium	Fixed
N6	Token compatibility reminder	Others	Suggestion	Acknowledged
N7	Missing boundary condition checks	Design Logic Audit	Medium	Acknowledged
N8	Improper use of symbols in setTraitConstraints function	Design Logic Audit	High	Fixed
N9	Missing event records	Others	Suggestion	Fixed
N10	Authority transfer enhancement	Others	Suggestion	Acknowledged
N11	Preemptive Initialization	Reordering Vulnerability	Suggestion	Acknowledged
N12	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/flooringlab/bmp-erc721

commit: 363a72c6372a9d226153c04715e6b10f369c98b7

Fixed Version:

https://github.com/flooringlab/bmp-erc721

commit: 2310685e84db3e24465d928e99e9a3cf7ebfdc0f

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

BitmapPunks			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Payable	-
name	Public	-	-
symbol	Public	-	-
revealNFT	Public	Can Modify State	checkRevealable
concealNFT	Public	Can Modify State	-
mint	Public	Payable	onlyOwnerOrRoles checkAndUpdateTotalMinted

BitmapPunks			
setExchangeNFTFeeRate	Public	Can Modify State	onlyOwnerOrRoles
setRevealable	Public	Can Modify State	onlyOwnerOrRoles
_guardInitializeOwner	Internal	-	-
_authorizeUpgrade	Internal	Can Modify State	onlyOwnerOrRoles
_afterConsecutiveMints	Internal	Can Modify State	-

BitmapPunks721			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BitmapBT404Mirror
initialize	Public	Can Modify State	-
tokenURI	Public	-	-
exchange	Public	Can Modify State	verifyOracleSignature
exchangeByMigrator	Public	Can Modify State	onlyOwnerOrRoles
addTokenBatch	Public	Can Modify State	onlyOwnerOrRoles
setOracle	Public	Can Modify State	onlyOwner
setOracleSigBlockRange	Public	Can Modify State	onlyOwner
_checkOwnerOrRoles	Internal	-	-
_useOracleNonce	Internal	Can Modify State	-
_hashExchangeData	Internal	-	-
_checkOracleSignature	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	onlyOwnerOrRoles
<Fallback>	External	Payable	bt404NFTFallback

BitmapPunksMigration			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
migrateToken	Public	Can Modify State	onlyOwner
migratePunks	Public	Can Modify State	onlyOwner
withdraw	Public	Can Modify State	onlyOwner
withdraw	Public	Can Modify State	onlyOwner

BitmapTraits			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

BitmapBT404			
Function Name	Visibility	Mutability	Modifiers
tokenURI	Public	-	-
_mint	Internal	Can Modify State	-
_afterConsecutiveMints	Internal	Can Modify State	-

BitmapBT404Mirror			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BT404Mirror
<Fallback>	External	Payable	bt404NFTFallback bitmapFallback

BT404			
Function Name	Visibility	Mutability	Modifiers
_getBT404Storage	Internal	-	-
_initializeBT404	Internal	Can Modify State	-
_unit	Internal	-	-
name	Public	-	-
symbol	Public	-	-
tokenURI	Public	-	-
decimals	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-
allowance	Public	-	-
approve	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
_transfer	Internal	Can Modify State	-
_revealNFT	Internal	Can Modify State	-
_transferFromNFT	Internal	Can Modify State	-
_exchangeNFT	Internal	Can Modify State	-
_payExchangeFee	Internal	Can Modify State	-
_pullFeeForTwo	Internal	Can Modify State	-
_mintNFT	Internal	Can Modify State	-
_burnNFT	Internal	Can Modify State	-

BT404			
_approve	Internal	Can Modify State	-
_getAux	Internal	-	-
_setAux	Internal	Can Modify State	-
getLockedBalance	Public	-	-
_setExchangeNFTFeeRate	Internal	Can Modify State	-
_setListMarketFeeRate	Internal	Can Modify State	-
_addressData	Internal	Can Modify State	-
_registerAndResolveAlias	Internal	Can Modify State	-
mirrorERC721	Public	-	-
_totalNFTSupply	Internal	-	-
_balanceOfNFT	Internal	-	-
_ownerAt	Internal	-	-
_ownerOf	Internal	-	-
_exists	Internal	-	-
_getApproved	Internal	-	-
_approveNFT	Internal	Can Modify State	-
_removeNFTApproval	Internal	Can Modify State	-
_setApprovalForAll	Internal	Can Modify State	-
_setNFTLockState	Internal	Can Modify State	-
_ownedIds	Internal	-	-
_offerForSale	Internal	Can Modify State	-
_acceptOffer	Internal	Can Modify State	-

BT404			
_cancelOffer	Internal	Can Modify State	-
_clearNFTOffer	Internal	Can Modify State	-
_bidForBuy	Internal	Can Modify State	-
_acceptBid	Internal	Can Modify State	-
_cancelBid	Internal	Can Modify State	-
_transferToken	Private	Can Modify State	-
<Fallback>	External	Payable	bt404Fallback
<Receive Ether>	External	Payable	-
_ownershipIndex	Internal	-	-
_ownedIndex	Internal	-	-
_getOwnedIndexOf	Internal	-	-
_delNFTAt	Internal	Can Modify State	-
_totalSupplyOverflows	Internal	-	-
_packedLogsMalloc	Internal	-	-
_packedLogsSet	Internal	-	-
_packedLogsAppend	Internal	-	-
_packedLogsSend	Internal	Can Modify State	-
_calldataload	Internal	-	-
_calldatacopyArray	Private	-	-
_calldatacopyOrders	Private	-	-
_return	Internal	-	-
_zeroFloorSub	Internal	-	-

BT404			
_min	Internal	-	-
_max	Internal	-	-
_toUint	Internal	-	-
_get	Internal	-	-
_set	Internal	Can Modify State	-
_setOwnerAliasAndOwnedIndex	Internal	Can Modify State	-

BT404Mirror			
Function Name	Visibility	Mutability	Modifiers
_getBT404NFTStorage	Internal	-	-
<Constructor>	Public	Can Modify State	-
_initializeBT404Mirror	Internal	Can Modify State	-
name	Public	-	-
symbol	Public	-	-
tokenURI	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-
ownerOf	Public	-	-
ownerAt	Public	-	-
approve	Public	Can Modify State	-
getApproved	Public	-	-
setApprovalForAll	Public	Can Modify State	-
isApprovedForAll	Public	-	-

BT404Mirror			
ownedIds	Public	-	-
lockedIds	Public	-	-
transferFrom	Public	Can Modify State	-
safeTransferFrom	Public	Can Modify State	-
safeTransferFrom	Public	Can Modify State	-
updateLockState	Public	Can Modify State	-
_exchange	Internal	Can Modify State	-
offerForSale	Public	Can Modify State	nonReentrant
acceptOffer	Public	Payable	nonReentrant
cancelOffer	Public	Can Modify State	nonReentrant
bidForBuy	Public	Payable	nonReentrant
acceptBid	Public	Can Modify State	nonReentrant
cancelBid	Public	Can Modify State	nonReentrant
supportsInterface	Public	-	-
owner	Public	-	-
pullOwner	Public	Can Modify State	-
baseERC20	Public	-	-
<Fallback>	External	Payable	bt404NFTFallback
<Receive Ether>	External	Payable	-
_ownedIds	Private	-	-
_readString	Private	-	-
_readWord	Internal	-	-

BT404Mirror			
_callBaseRetWord	Private	Can Modify State	-
_calldataload	Internal	-	-
_hasCode	Private	-	-
_checkOnERC721Received	Private	Can Modify State	-

Bitmap721Template			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BitmapBT404Mirror
_checkOwnerOrRoles	Internal	-	-
_authorizeUpgrade	Internal	Can Modify State	onlyOwnerOrRoles
initialize	Public	Can Modify State	-
tokenURI	Public	-	-

BitmapLayersUpgradeable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
initialize	Public	Can Modify State	initializer

BitmapTraitsUpgradeable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
initialize	Public	Can Modify State	initializer

PaletteRegistry			
Function Name	Visibility	Mutability	Modifiers
_getPaletteStorage	Internal	-	-
createPalette	Public	Can Modify State	-
getPalettes	Public	-	-
getPalette	Public	-	-
lastPaletteId	Public	-	-
_setPalette	Internal	Can Modify State	-
_getPaletteColors	Internal	-	-

PixelLayerRegistry			
Function Name	Visibility	Mutability	Modifiers
_getLayerStorage	Internal	-	-
createLayer	Public	Can Modify State	-
createLayer	Public	Can Modify State	-
createLayer	Public	Can Modify State	-
createLayer	Public	Can Modify State	-
createPixelData	Public	Can Modify State	-
sortLayersByZIndex	Public	-	-
getLayers	Public	-	-
getLayer	Public	-	-
getPixelData	Public	-	-
lastLayerId	Public	-	-
_sortLayersByZIndex	Internal	-	-

PixelLayerRegistry			
_createLayer	Internal	Can Modify State	-
_createPixelData	Internal	Can Modify State	-
_getPixelDataRef	Internal	-	-
_checkLayerData	Internal	-	-
_checkLayerRegion	Internal	-	-
_isValidBitsPerPixel	Internal	-	-
_getLayerInner	Internal	-	-
_loadMemoryArray	Internal	-	-
_storeMemoryArray	Internal	-	-

PixelLayerResolver			
Function Name	Visibility	Mutability	Modifiers
createPaletteAndLayer	Public	Can Modify State	-
genBMPFromLayers	Public	-	-
genSVGFromLayers	Public	-	-
compositeLayers	Public	-	-
_compositeLayerWithPalette	Internal	-	-
_compositeActiveRegion	Internal	-	-
_compositeLayerBG	Internal	-	-
_compositeLayerRectBG	Internal	-	-
_blendBGRA	Internal	-	-

TraitRegistry			
Function Name	Visibility	Mutability	Modifiers
_getTraitStorage	Internal	-	-
_initializeTraitRegistry	Internal	Can Modify State	-
createTraits	Public	Can Modify State	-
createTraitPool	Public	Can Modify State	-
createTraitsAndPool	Public	Can Modify State	-
createTraitType	Public	Can Modify State	-
setTraitConstraints	Public	Can Modify State	-
setTraitPoolConstraints	Public	Can Modify State	-
setTraitWeightInPool	Public	Can Modify State	-
registerCollection	Public	Can Modify State	-
setCollectionImageFormat	Public	Can Modify State	-
transferCollectionOwnership	Public	Can Modify State	-
lastTraitId	Public	-	-
getTraits	Public	-	-
collectionConfig	Public	-	-
getTraitPool	Public	-	-
getTraitType	Public	-	-
traitToTraitRelation	Public	-	-
traitToPoolRelation	Public	-	-
poolToPoolRelation	Public	-	-
getCollectionTraitTypeCount	Public	-	-

TraitRegistry

getCollectionTraitTypeCount	Public	-	-
getImageURIOf	Public	-	-
generateRandomTraits	Public	-	-
generateRandomTraitsFor	Public	-	-
getAttibutesJson	Public	-	-
getAttibutesJson	Public	-	-
_getRandomTrait	Internal	-	-
_filterTraits	Internal	-	-
_isMatchingConstraint	Internal	-	-
_isLimited	Internal	-	-
_listContains	Internal	-	-
_getTraitLayers	Internal	-	-
_setBitmap	Internal	Can Modify State	-
_loadCalldataArray	Internal	-	-
_loadMemoryArray	Internal	-	-
_storeMemoryArray	Internal	-	-

TraitsMetadata

Function Name	Visibility	Mutability	Modifiers
_getTraitsMetadataStorage	Internal	-	-
_initializeTraitsMetadata	Internal	Can Modify State	-
tokenTraits	Public	-	-
traitRegistry	Public	-	-

TraitsMetadata			
_getTokenAttributesAndImage	Internal	-	-
_addTokenBatch	Internal	Can Modify State	-
_findTokenBatch	Internal	-	-
_getTraitRandomSeeds	Internal	-	-
_readStringByArray	Private	-	-

4.3 Vulnerability Summary

[N1] [Critical] Lack of fee settlement when minting NFTs

Category: Design Logic Audit

Content

In the BitmapBT404 contract, the _mint function is used for NFT minting operations. However, within the function, before minting NFTs for users, the _pullFeeForTwo function is not called to settle the claimable fees of the NFTs previously seeded by users. This will cause these newly minted NFTs to be able to directly participate in the fee calculation, allowing users to claim more fees than expected, and may prevent other seeding users from normally claiming rewards.

Code Location:

src/bt404/BitmappBT404.sol#L35-94

```
    function _mint(address to, uint256 amount) internal virtual {
         ...
     }
```

Solution

It is recommended that when minting new NFTs for users, the _pullFeeForTwo function should be called first to settle the unclaimed fees.

Status

Fixed

[N2] [High] Missing update numExchangableNFT when minting NFTs

Category: Design Logic Audit

Content

In the BitmapBT404 contract, when the _mint function is called to mint NFTs for users, it doesn't check whether the user's operatorApprovals for the contract address is true (that is, check the value of $.operatorApprovals[address(this)][to]), resulting in the new minted NFT quantity not being added to $.numExchangableNFT. This will cause other functions of the contract to be affected due to $.numExchangableNFT.

Code Location:

src/bt404/BitmappBT404.sol#L35-94

```
    function _mint(address to, uint256 amount) internal virtual {
        ...
     }
```

Solution

It is recommended to check whether the value of `$.operatorApprovals[address(this)][owner]` is true after minting NFTs for users, and then update the value of $.numExchangableNFT.

Status

Fixed

[N3] [Medium] Missing chainID check in the signature verification

Category: Replay Vulnerability

Content

In the BitmapPunks721 contract, the exchange function allows users to input data such as the tokenId of the specified NFT and custom fees. After verification by the verifyOracleSignature function, the NFT exchange operation is carried out. However, when hashing the data that needs to be signed, the chainId is not in included the hash. This means that if the project is deployed on multiple chains, the signature may be maliciously replayed by attackers on another chain, resulting in unexpected fee information being passed in.

Code Location:

src/bt404/BitmapPunks721.sol#L89&L168

```
    function exchange(uint256 idX, uint256 idY, uint32 feeRate, bytes calldata
signature)
        public
        verifyOracleSignature(_hashExchangeData(idY, feeRate, msg.sender), signature)
        returns (uint256 exchangeFee)
    {
        return _exchange(idX, idY, feeRate);
    }

    function _checkOracleSignature(bytes32 msgHash, bytes calldata signature) internal
{
        ...

        msgHash = keccak256(abi.encodePacked(msgHash, blockNumber, nonce));

        require(
            SignatureCheckerLib.isValidSignatureNow(oracle, msgHash, r, vs),
            InvalidOracleSignature()
        );
    }
```

Solution

If the project plans to be deployed on multiple chains, it’s recommended to add the chainId in the hash calculation of the signature message.

Status

Fixed

[N4] [High] Incorrect condition for numExchangableNFT update in the _exchangeNFT function

Category: Design Logic Audit

Content

In the BT404 contract, the _exchangeNFT function is used for the exchange operation of two NFTs. It transfers the NFT corresponding to idX from its owner's address to the owner of the NFT corresponding to idY, and transfers the NFT corresponding to idY from its owner's address to the owner of the NFT corresponding to idX and locks it. When the NFT corresponding to idY is not from the burn pool and the user's operatorApprovals for the contract address is true, the value of $.numExchangableNFT is updated by decrementing. However, under normal circumstances, when an NFT is transferred to the burn pool or the NFT obtained by the user needs to be locked, the value of $.numExchangableNFT will be decremented. In the _exchangeNFT function, the NFT obtained by the user will be locked after the transfer regardless of whether it comes from the burn pool. So the condition check !exchangeBurned here is redundant.

Let's use an example to illustrate this situation:

1.Suppose a user holds an NFT with a tokenId of 1, and there is an NFT in the burn pool with a tokenId of 2. The user's operatorApprovals for the contract is true, and at this time, $.numExchangableNFT is equal to 1.

2.The user calls the exchange function to perform the NFT exchange operation. At this time, the NFT with tokenId of 2 will be transferred from the burn pool to the user and locked, and the NFT with tokenId of 1 will be transferred to the burn pool. Since one of the two NFTs is locked and the other is in the burn pool, the value of $.numExchangableNFT should be 0. However, because exchangeBurned is true, the update of $.numExchangableNFT in the _exchangeNFT function is skipped, resulting in the value of $.numExchangableNFT still being equal to 1.

Code Location:

src/bt404/BT404.sol#L708

```
    function _exchangeNFT(uint256 idX, uint256 idY, uint256 feeRate, address
msgSender)
        internal
        virtual
        returns (address, address, uint256)
    {
        ...

        bool exchangeBurned = _get($.oo, _ownershipIndex(idY)) ==
_ADDRESS_ALIAS_BURNED_POOL;
        {
            mapping(address => Uint256Ref) storage thisOperatorApprovals =
                $.operatorApprovals[address(this)];
            /// Only Burned or Approved NFT can be exchanged.
            if (!exchangeBurned && thisOperatorApprovals[y].value == 0) {
                revert ApprovalCallerNotOwnerNorApproved();
            }
            // lock 1 NFT
            if (!exchangeBurned && thisOperatorApprovals[x].value != 0) {
                unchecked {
                    --$.numExchangableNFT;
                }
            }
        }

        ...
    }
```

Solution

It is recommended to remove the !exchangeBurned in the conditional check when updating $.numExchangableNFT.

Status

Fixed

[N5] [Medium] Pseudo-random risk

Category: Block data Dependence Vulnerability

Content

In the BT404 contract, the _revealNFT function is used to enable users to take out a random NFT from the burn pool. The obtained random TokenId is calculated using block.prevrandao, the user's address, and the number of NFTs the user currently holds. Unfortunately, these parameters can be controlled or are already known. As a result, the outcome is predictable, which allows malicious users to obtain high - value NFTs by predicting the random number.

Code Location:

src/bt404/BT404.sol#L540

```
    function _revealNFT(address owner, uint256 nftAmount) internal virtual {
        ...

        do {
            uint256 randomIndex = uint256(
                keccak256(abi.encodePacked(block.prevrandao, owner, toIndex))
            ) % burnedPoolSize;

            uint256 id = _get($.burnedPool, randomIndex);
            if (randomIndex != (--burnedPoolSize)) {
                uint32 lastId = _get($.burnedPool, burnedPoolSize);
                _set($.burnedPool, randomIndex, lastId);
                _setOwnerAliasAndOwnedIndex(
                    oo, lastId, _ADDRESS_ALIAS_BURNED_POOL, uint32(randomIndex)
                );
            }

            _set(owned, toIndex, uint32(id));
            _setOwnerAliasAndOwnedIndex(oo, id, senderAlias, uint32(toIndex++));
            _packedLogsAppend(packedLogs, id);
        } while (toIndex != toEnd);

        ...
    }
```

Solution

Using Chainlink VRF is the best practice for using random numbers on the blockchain, but it comes with a relatively high cost.

Another viable solution is to first identify a future block (for example, 4 epochs in advance). When the user is allowed to claim the NFT upon reaching that block, the block.prevrandao of that future block will be used to calculate the tokenId. Since the block.prevrandao comes from the specified future block, it can better meet the requirements for randomness.

Status

Fixed; Updated: Use a random number parameter that has passed signature verification to calculate the tokenId.

[N6] [Suggestion] Token compatibility reminder

Category: Others

Content

In the BT404 contract, when making a quoted transaction for an NFT, the _transferToken function is used to transfer the specified ERC20 tokens. However, if this function uses SafeTransferLib.safeTransferFrom for the transfer, it does not check the difference between the balance before and after the transfer to the recipient address against the transferred amount. If the tokens to be transferred are deflationary tokens, then the actual number of tokens received will not match the number of tokens recorded in the contract.

Code Location:

src/bt404/BT404.sol#L1432

```
      function _bidForBuy(address msgSender, NFTOrder[] memory orders) internal {
        ...

        for (uint256 i; i < orders.length;) {
            ...

            {
                ...

                // Refund exist bid.(Prevent Reentrancy externally)
                _transferToken(bid.bidToken, address(this), msgSender, bid.tokens);
                // Receive new bid funds.
                _transferToken(token, msgSender, address(this), tokenUnits);
                if (token == address(0)) nativeBidTokens += tokenUnits;
            }

            unchecked {
                ++i;
            }
        }

        ...
    }

    function _transferToken(address token, address from, address to, uint256 amount)
private {
        if (token == address(0)) {
            ...
        } else if (token == address(this)) {
            ...
        } else {
            if (from == address(this)) {
                SafeTransferLib.safeTransfer(token, to, amount);
            } else {
                SafeTransferLib.safeTransferFrom(token, from, to, amount);
            }
        }
    }
```

Solution

It is recommended to use the difference between the recipient address's balance before and after the transfer to record the user's actual transfer amount.

Status

Acknowledged; The project team responded: Deflationary tokens will not be considered, and only tokens officially supported will be displayed on the front-end.

[N7] [Medium] Missing boundary condition checks

Category: Design Logic Audit

Content

1.In the TraitRegistry contract, when a user calls the createTraits function to create traits, they need to set the layerIds of the corresponding layers of the trait. However, in this function, it does not check whether each of the incoming layerId is less than $.lastLayerId in the PixelLayerRegistry contract. This may lead to the situation where a user could set the id of a layer that has not been created yet for the trait.

Code Location:

src/TraitRegistry.sol#L203

```
    function createTraits(TraitParam[] calldata traitList)
         public
         returns (uint256[] memory traitIds)
     {
         ...

         for (uint256 i = 0; i < length; ++i) {
             ...

             traitIds[i] = traitId;
             trait.layerIds = param.layerIds;

             ...
         }

         ...
     }
```

2.In the TraitRegistry contract, when a user calls the createTraitPool function to create trait pools, they need to set the traits owned by this pool. However, in this function, it does not check whether each of the incoming traitId is not greater than $.lastTraitId. This may lead to the situation where a user could set the id of a trait that has not been created yet for the pool.

Code Location:

src/TraitRegistry.sol#L227

```
   function createTraitPool(address collection, uint256[] memory traitIds)
        public
        returns (uint256 poolIndex)
    {
        ...

        for (uint256 k = 0; k < count; ++k) {
            p.traitIds.set(k, _loadMemoryArray(traitIds, k).toUint32());
        }
        ...
    }
```

3.In the TraitRegistry contract, when a user calls the createTraitType function to create trait types, they need to set the pools owned by this type. However, in this function, it does not check whether each of the incoming poolIndex is less than config.poolCount. This may lead to the situation where a user could set the index of a pool that has not been created yet for the trait type.

Code Location:

src/TraitRegistry.sol#L263

```
   function createTraitType(address collection, TraitTypeParam calldata param)
        public
        returns (uint256 traitTypeIndex)
    {
        ...

        for (uint256 k = 0; k < poolCount; ++k) {
            st.poolIndexes.set(k, _loadCalldataArray(param.poolIndexes,
k).toUint16());
        }
        st.poolCount = uint16(poolCount);

        ...
    }
```

4.In the TraitRegistry contract, when a user calls the setTraitWeightInPool function, it sets the weight of each trait in the trait pool. However, in this function, it does not check whether the pool corresponding to the specified poolIndex has been created. At the same time, it does not check whether each of the incoming traitIds is within the range of the traitIds in the trait pool (i.e., pool.traitIds). This means that users may set weights for traits that are not yet in the pool.

Code Location:

src/TraitRegistry.sol#L227

```
function setTraitWeightInPool(
     address collection,
     uint256 poolIndex,
     uint256[] calldata traitIds,
     uint256[] calldata weights
 ) public {
     ...

     TraitPool storage pool = config.traitPools[poolIndex.toUint16()];

     uint256 length = traitIds.length;
     require(length == weights.length, TraitsAndWeightsNotMatch());

     for (uint256 i = 0; i < length; ++i) {
         uint16 weight = _loadCalldataArray(weights, i).toUint16();
         require(weight > 0, InvalidTraitWeight());
         pool.traitWeights.set(_loadCalldataArray(traitIds, i).toUint32(), weight);
     }

     emit TraitPoolWeightSet(collection, poolIndex, traitIds, weights);
 }
```

5.In the PixelLayerRegistry contract, when a user calls the createLayer function to create a layer, they need to set the palette corresponding to this layer. However, in this function, it doesn't check whether the incoming paletteId is not greater than $.lastPaletteId in the PaletteRegistry contract. This may cause users to set an uncreated palette for the layer.

Code Location:

src/PixelLayerRegistry.sol#L146

```
function createLayer(
     LayerParam calldata layerParam,
     RectRegion memory regionParam,
     uint256 bgcolorOrIndex,
     bytes calldata pixels,
        uint256 paletteId
    ) public returns (uint256) {
        ...

        require(paletteId > 0, InvalidPaletteId());

        ...

        return _createLayer(layerParam, regionParam, bgcolorOrIndex, pixelRefId,
paletteId);
    }


   function _createLayer(
        LayerParam calldata layerParam,
        RectRegion memory regionParam,
        uint256 bgcolorOrIndex,
        uint256 pixelRefId,
        uint256 paletteId
    ) internal returns (uint256 layerId) {
        {
            ...

            layer.paletteId = paletteId.toUint32();

            ...
        }

        ...
    }
```

Solution

It is recommended to add appropriate boundary checks in the above functions to prevent unexpected situations.

Status

Acknowledged

[N8] [High] Improper use of symbols in setTraitConstraints function

Category: Design Logic Audit

Content

In the TraitRegistry contract, when the setTraitConstraints and setTraitPoolConstraints functions are called, the trait constraint lists and pool constraint lists corresponding to traits and trait pools are set. The _setBitmap function is used to add data to these lists. However, in the _setBitmap function, when new data is added, it returns the number of newly added items instead of the result of adding to the original number of items in the list. Moreover, when updating the constraint count in the setTraitConstraints and setTraitPoolConstraints functions, direct equal sign assignment is used instead of using += for accumulation. This results in incorrect changes to the count of constraints during the update.

Code Location:

src/TraitRegistry.sol

```
   function setTraitConstraints(
        address collection,
        uint256 baseTraitId,
        uint256[] calldata traitBlockList,
        uint256[] calldata traitAllowList,
        uint256[] calldata poolBlockList,
        uint256[] calldata poolAllowList
    ) public {
        {
            CollectionConfig storage config =
_getTraitStorage().collectionConfigs[collection];
            require(config.owner == msg.sender, NotOwnerOfCollection());

            TraitConstraint storage constraint =
config.traitConstraints[baseTraitId.toUint32()];

            uint256 traitBlockCount = _setBitmap(constraint.traitBlockList,
traitBlockList);
            uint256 traitAllowCount = _setBitmap(constraint.traitAllowList,
traitAllowList);
            uint256 poolBlockCount = _setBitmap(constraint.poolBlockList,
poolBlockList);
            uint256 poolAllowCount = _setBitmap(constraint.poolAllowList,
poolAllowList);

            constraint.traitBlockCount = uint32(traitBlockCount);
            constraint.traitAllowCount = uint32(traitAllowCount);
            constraint.poolBlockCount = uint16(poolBlockCount);
            constraint.poolAllowCount = uint16(poolAllowCount);
        }

        emit TraitConstraintSet(
            collection, baseTraitId, traitBlockList, traitAllowList, poolBlockList,
poolAllowList
        );
    }

    function setTraitPoolConstraints(
        address collection,
        uint256 basePoolIndex,
        uint256[] calldata poolBlockList,
        uint256[] calldata poolAllowList,
        uint256[] calldata traitBlockList,
        uint256[] calldata traitAllowList
    ) public {
        CollectionConfig storage config =
_getTraitStorage().collectionConfigs[collection];
        require(config.owner == msg.sender, NotOwnerOfCollection());

        TraitPool storage p = config.traitPools[basePoolIndex.toUint16()];

        {
            uint256 poolBlockCount = _setBitmap(p.poolBlockList, poolBlockList);
            uint256 poolAllowCount = _setBitmap(p.poolAllowList, poolAllowList);
            uint256 traitBlockCount = _setBitmap(p.traitBlockList, traitBlockList);
            uint256 traitAllowCount = _setBitmap(p.traitAllowList, traitAllowList);

            p.poolBlockCount = uint16(poolBlockCount);
            p.poolAllowCount = uint16(poolAllowCount);
            p.traitBlockCount = uint32(traitBlockCount);
            p.traitAllowCount = uint32(traitAllowCount);
        }

        emit TraitPoolConstraintSet(
            collection, basePoolIndex, poolBlockList, poolAllowList, traitBlockList,
traitAllowList
        );
    }

   function _setBitmap(Bitmap storage map, uint256[] calldata vals)
        internal
        returns (uint256 changedCount)
    {
        unchecked {
            uint256 length = vals.length;
            for (uint256 i = 0; i < length; ++i) {
                uint256 val = _loadCalldataArray(vals, i);
                if (!map.get(val)) {
                    map.set(val);
                    ++changedCount;
                }
            }
        }
```

Solution

It is recommended to change the `=` to `+=` when updating the constraint count in these two functions.

Status

Fixed

[N9] [Suggestion] Missing event records

Category: Others

Content

1.In the BitmapPunks contract, the migration manager role can set whether to enable the reveal through the setRevealable function. However, no event logging is performed.

Code Location:

src/bitmap-punk/BitmapPunks.sol#L84-86

```
   function setRevealable(bool _revealable) public
onlyOwnerOrRoles(_MIGRATION_MANAGER_ROLE) {
        revealable = _revealable;
    }
```

2.In the BitmapPunks721 contract, the owner role can set the oracle address and oracleSigBlockRange. However, no event logging is performed.

Code Location:

src/bitmap-punk/BitmapPunks721.sol#L123-129

```
    function setOracle(address oracle_) public onlyOwner {
        oracle = oracle_;
    }

    function setOracleSigBlockRange(uint48 oracleSigBlockRange_) public onlyOwner {
        oracleSigBlockRange = oracleSigBlockRange_;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N10] [Suggestion] Authority transfer enhancement

Category: Others

Content

There is no pending and accept mechanism for authority transfer to avoid loss of authority. If the new owner is incorrectly set, the permission will be lost.

Code Location:

src/TraitRegistry.sol

```
    function transferCollectionOwnership(address collection, address newOwner) public
{
        CollectionConfig storage config =
_getTraitStorage().collectionConfigs[collection];
        require(config.owner == msg.sender, NotOwnerOfCollection());

        config.owner = newOwner;

        emit CollectionOwnershipTransferred(collection, msg.sender, newOwner);
    }
```

Solution

It is recommended to have a pending and accept operation when transferring minting authority to avoid losing authority.

Status

Acknowledged

[N11] [Suggestion] Preemptive Initialization

Category: Reordering Vulnerability

Content

By calling the initialize function to initialize the contract, there is a potential issue that malicious attackers preemptively call the initialize function to initialize.

Code location:

src/bitmap-punk/BitmapPunks.sol#L46-53

src/bitmap-punk/BitmapPunks721.sol#L46-52

src/bitmap-punk/BitmapPunksMigration.sol#L35-37

```
function initialize(...) public payable {
    ...
}
```

src/example/BitmapLayersUpgradeable.sol#L16-18

src/example/BitmapTraitsUpgradeable.sol#L16-19

```
function initialize() public initializer {
     __Ownable_init(tx.origin);
 }
```

Solution

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Acknowledged; The project team responded: During the deployment process, we will try our best to ensure that the contract is deployed and initialized simultaneously. Currently, we are using Foundry script.

[N12] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the BitmapPunks contract, the mint manager role can mint tokens and NFTs for any specified address through the mint function. If this role is set to an EOA address and its permission is compromised, it could affect the normal operation of the project.

Code Location:

src/bitmap-punk/BitmapPunks.sol#L71-78

```
   function mint(address to, uint256 nftAmount)
        public
        payable
        onlyOwnerOrRoles(_MINT_MANAGER_ROLE)
        checkAndUpdateTotalMinted(nftAmount)
    {
        _mint(to, nftAmount * _unit());
    }
```

2.In the BitmapPunks721 contract, the owner role can set the oracle address and oracleSigBlockRange through the setOracle and setOracleSigBlockRange functions. In addition, both the owner role and the migration manager role can set the seeds for specified NFTs and exchange any two NFTs by calling the addTokenBatch and exchangeByMigrator functions. If the private keys of these roles are leaked, it will cause the loss of users' funds.

Code Location:

src/bitmap-punk/BitmapPunks721.sol#L100-129

```
    function exchangeByMigrator(uint256 idX, uint256 idY)
        public
        onlyOwnerOrRoles(_MIGRATION_MANAGER_ROLE)
        returns (uint256 exchangeFee)
    {
        return _exchange(idX, idY, 0);
    }

    function addTokenBatch(uint256 fromTokenId, uint256 toTokenId, uint256 seed)
        public
        onlyOwnerOrRoles(_MIGRATION_MANAGER_ROLE)
    {
        _addTokenBatch(fromTokenId, toTokenId, seed);
    }

    function setOracle(address oracle_) public onlyOwner {
        oracle = oracle_;
    }

    function setOracleSigBlockRange(uint48 oracleSigBlockRange_) public onlyOwner {
        oracleSigBlockRange = oracleSigBlockRange_;
    }
```

3.In the BitmapPunksMigration contract, the owner role can transfer out any assets in the contract, including tokens and NFTs. If this role is set to an EOA address and its permission is compromised, it will cause the loss of the contract’s funds.

Code Location:

src/bitmap-punk/BitmapPunksMigration.sol

```
   function migrateToken(address token, address recipient, uint256 amount) public
onlyOwner {
        IBT404(token).transfer(recipient, amount);
    }

    function migratePunks(
        address collection,
        address recipient,
        uint256[] calldata ids,
        uint256[] calldata seeds
    ) public onlyOwner {
        ...
    }

    function withdraw(address token, uint256 amount) public onlyOwner {
        ...
    }

    function withdraw(address collection, uint256[] calldata ids) public onlyOwner {
        ...
    }
```

4.The UUPSUpgradeable MANAGER_ROLE relevant authority can upgrade the contract, leading to the risk of over-privileged in this role.

Solution

In the short term, transferring the ownership of core roles to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002504100002	SlowMist Security Team	2025.03.20 - 2025.04.10	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 3 high risks, 3 medium risks, 1 low risks and 4 suggestion. All the findings were fixed or acknowledged. Since the project has not yet been deployed to the mainnet and the permissions of the core roles have not yet been transferred, the risk level reported is temporarily medium.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist