ERC721v2 Contract Audit

ERC721v2 Preset Contract

Internal Auditors	Mark Art, Ryan Teoh
Assessment Date	February 21, 2025
Final Report	February 26, 2025
Status	Completed
GitHub PR	https://github.com/immutable/contracts/pull/265

Previous audits: https://github.com/immutable/contracts/blob/main/audits/202309-threat-model-preset-erc721.md

Contents

Introduction

The newly introduced ImmutableERC721V2 contract and associated ERC 721 V2 contracts represent a significant advancement in optimizing gas consumption, particularly within scenarios where mint by quantity is used with large numbers of tokens, and when the total supply of a collection is large. This enhancement directly addresses the limitations of its predecessor, the SAR-48 ERC721 Preset Contract, which encountered scalability challenges when dealing with a sufficiently large number of tokens. Consequently, the ImmutableERC721V2 contracts ensure a more sustainable and efficient operation as the token ecosystem expands, thereby providing a robust foundation for projects that anticipate significant growth in their token supply.

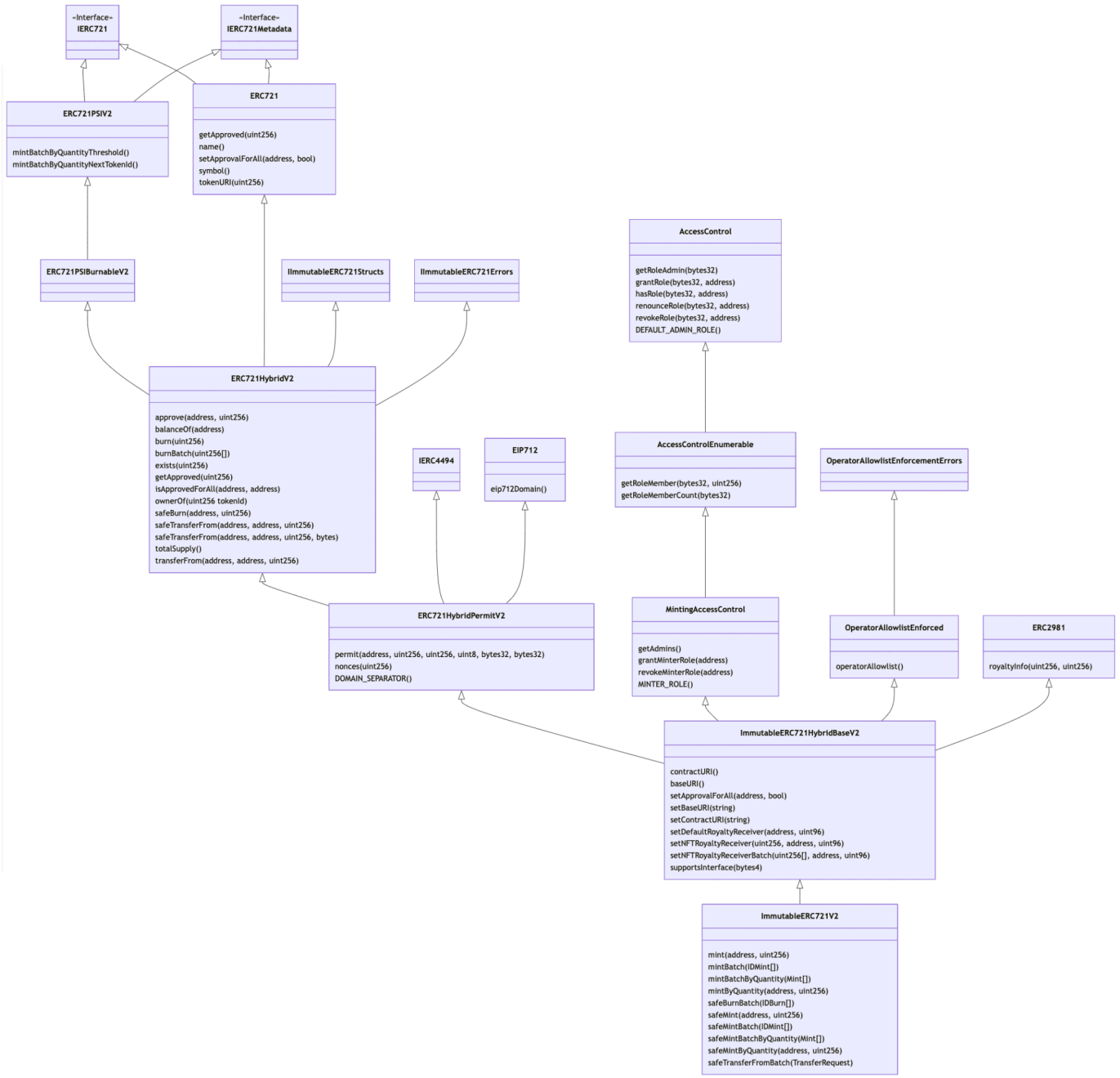
The significant change that drives the improved performance of ImmutableERC721V2 is ERC721PsiV2 and ERC721PsiBurnableV2. ERC721Psi uses large arrays of bits. If a large number of NFTs were minted via a single mint by quantity, then a large array is generated. A scan of this large array is then needed each time balanceOf is called. This in turn causes significant performance issues. ERC721PsiV2 is especially designed such that default token ownership is at a known location. This means that an array scan is not required. The result is that the performance does not increase as a result of contract usage.

Scope

Commit: 2606a379573b892428254b83660b4bc91ed6e173

ERC-721v2 Architecture

The diagram on the following page shows the ImmutableERC721V2 contract and related contracts. The implementation location of external methods that are available in the ImmutableERC721V2 contract are shown. To limit the size of the diagram, Open Zeppelin contracts that do not define external functions that are entry point functions for the ImmutableERC721V2 contract are not shown.



External Functions

State-Changing Functions

Function Name	Function Selector	Access Control
approve(address,uint256)	095ea7b3	msg.sender must be NFT owner or approvedForAll the owner's NFTs.
burn(uint256)	42966c68	For all NFTs, msg.sender must be owner or approved.
burnBatch(uint256[])	e4623c1b	For all NFTs, msg.sender must be owner or approved.

grantMinterRole(address)	3dd1eb61	Only DEFAULT_ADMIN_ROLE
grantRole(bytes32,address)	2f2ff15d	Only role admin for role
mint(address,uint256)	40c10f19	Only MINTER_ROLE
mintBatch((address,uint256[])[])	9e2641a3	Only MINTER_ROLE
mintBatchByQuantity((address,uint256)[])	e1927859	Only MINTER_ROLE
mintByQuantity(address,uint256)	eea98768	Only MINTER_ROLE
permit(address,uint256,uint256,bytes)	745a41bc	Signer of permit must be NFT owner or approved.
renounceRole(bytes32,address)	36568abe	msg.sender can only renounce their own roles.
revokeMinterRole(address)	69e2f0fb	Only DEFAULT_ADMIN_ROLE
revokeRole(bytes32,address)	d547741f	Only role admin for role
safeBurn(address,uint256)	9f15d700	msg.sender must be owner or approved.
safeBurnBatch((address,uint256[])[])	74f16a81	For all NFTs, msg.sender must be owner or approved.
safeMint(address,uint256)	a1448194	Only MINTER_ROLE
safeMintBatch((address,uint256[])[])	234af046	Only MINTER_ROLE
safeMintBatchByQuantity((address,uint256)[])	2d2c5a8b	Only MINTER_ROLE
safeMintByQuantity(address,uint256)	567e4075	Only MINTER_ROLE
safeTransferFrom(address,address,uint256)	42842e0e	msg.sender must be owner or approved.
safeTransferFrom(address,address,uint256,bytes)	b88d4fde	msg.sender must be owner or approved.
safeTransferFromBatch((address,address[],uint256[]))	008ad946	For all NFTs: msg.sender must be owner or approved.
setApprovalForAll(address,bool)	a22cb465	Msg.sender can only specify an operator for their own NFTs.
setBaseURI(string)	55f804b3	Only DEFAULT_ADMIN_ROLE
setContractURI(string)	938e3d7b	Only DEFAULT_ADMIN_ROLE
setDefaultRoyaltyReceiver(address,uint96)	885e7a08	Only DEFAULT_ADMIN_ROLE
setNFTRoyaltyReceiver(uint256,address,uint96)	439aed34	Only MINTER_ROLE
setNFTRoyaltyReceiverBatch(uint256[],address,uint96)	a7012816	Only MINTER_ROLE
transferFrom(address,address,uint256)	23b872dd	msg.sender must be owner or approved.

Non State-Changing Functions

Function Name	Function Selector
DEFAULT_ADMIN_ROLE()	a217fddf

DOMAIN_SEPARATOR()	3644e515
MINTER_ROLE()	d5391393
balanceOf(address)	70a08231
baseURI()	6c0360eb
contractURI()	e8a3d485
eip712Domain()	84b0196e
exists(uint256)	4f558e79
getAdmins()	31ae450b
getApproved(uint256)	081812fc
getRoleAdmin(bytes32)	248a9ca3
getRoleMember(bytes32,uint256)	9010d07c
getRoleMemberCount(bytes32)	ca15c873
hasRole(bytes32,address)	91d14854
isApprovedForAll(address,address)	e985e9c5
mintBatchByQuantityNextTokenId()	654167ea
mintBatchByQuantityThreshold()	315ed308
name()	06fdde03
nonces(uint256)	141a468c
operatorAllowlist()	29326f29
ownerOf(uint256)	6352211e
royaltyInfo(uint256,uint256)	2a55205a
supportsInterface(bytes4)	01ffc9a7
symbol()	95d89b41
tokenURI(uint256)	c87b56dd

ERC721PsiV2 Dependencies

Dependency / Import Path
@openzeppelin/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol
@openzeppelin/contracts/utils/Address.sol

@openzeppelin/contracts/utils/Context.sol
@openzeppelin/contracts/utils/Strings.sol
@openzeppelin/contracts/utils/introspection/ERC165.sol

Security Test Cases

TC-01: EIP-4494 Interface Compatibility

Result: PASS

Description: The contract correctly implements the EIP-4494 interface, enabling external contracts to identify supported features through interface detection.

Evidence: Interface signatures match specification in IERC4494.sol.

TC-02: Role-Based Access Control

Result: PASS

Description: All privileged operations are properly restricted to accounts with appropriate roles.

Evidence: Function modifiers consistently apply onlyRole checks with correct role parameters.

```
Unset
// Minting functions restricted to MINTER_ROLE
function mint(address to, uint256 tokenId) external override onlyRole(MINTER_ROLE) {
   _mint(to, tokenId);
}


function safeMint(address to, uint256 tokenId) external override onlyRole(MINTER_ROLE) {
   _safeMint(to, tokenId);
}


function mintByQuantity(address to, uint256 quantity) external override
onlyRole(MINTER_ROLE) {
   _mintByQuantity(to, quantity);
}


// Base URI updates restricted to DEFAULT_ADMIN_ROLE
function setBaseURI(string memory baseURI_) external override onlyRole(DEFAULT_ADMIN_ROLE)
{
   _setBaseURI(baseURI_);
}
```

Source

TC-03: NFT Burn Authorization

Result: PASS

Description: Burn operations verify the caller is either the token owner or an approved operator.

Evidence: Authorization checks implemented in burn(), burnBatch(), and safeBurn() functions.

```
Unset
function burn(uint256 tokenId) public virtual override {
    // Verify caller is authorized
    address owner = ownerOf(tokenId);
    if (
        msg.sender != owner &&
        !isApprovedForAll(owner, msg.sender) &&
        getApproved(tokenId) != msg.sender
    ) {
        revert NotOwnerOrApproved(msg.sender, tokenId);
    }

    _burn(tokenId);
}
```

Source

TC-04: EIP-712 Replay Protection

Result: PASS

Description: Permit signatures include domain separators and nonces to prevent replay attacks.

Evidence: Implementation leverages OpenZeppelin's audited EIP-712 libraries with correct parameter handling. Domain Separator and nonces are implemented to prevent replay attacks.

```
Unset
// Nonce tracking per token
mapping(uint256 => uint256) private _nonces;

// Nonce increment after use
function _useNonce(uint256 tokenId) internal virtual returns (uint256 current) {
    current = _nonces[tokenId];
    _nonces[tokenId] = current + 1;
}

// Nonce included in signature digest
bytes32 digest = _hashTypedDataV4(
    keccak256(
        abi.encode(
            _PERMIT_TYPEHASH,
```

```
            spender,
            tokenId,
            _useNonce(tokenId),
            deadline
        )
    )
);
```

Source

TC-05: Gas Consumption with Large Token Volumes

Result: PASS

Description: Operations remain gas-efficient even with high token counts.

Evidence: Performance tests demonstrate significant improvement over v1 implementation.

```
Unset
// Efficient token group storage
struct TokenGroup {
    address owner;
    uint256 burnBitmap;
}
// Optimized minting for batch operations
function _mint(address to, uint256 quantity) internal virtual {
    uint256 startTokenId = _nextTokenId();

    // Batch update for efficiency
    _balances[to] += quantity;
    _totalSupply += quantity;

    // Efficient storage of token groups
    uint256 updatedTokenId = startTokenId;
    for (uint256 i = 0; i < quantity; i += 256) {
        uint256 toMint = quantity - i;
        if (toMint > 256) {
            toMint = 256;
        }
        _owners[updatedTokenId / 256] = TokenGroup(to, 0);
        updatedTokenId += 256;
    }

    // Emit events
    for (uint256 i = 0; i < quantity; i++) {
        emit Transfer(address(0), to, startTokenId + i);
    }
}
```

Source

TC-06: Reentrancy Protection

Result: PASS

Description: External calls should be made after state changes to prevent reentrancy vulnerabilities.

Evidence : The ERC721 implementation from OpenZeppelin already includes reentrancy protection mechanisms. In critical functions like _transfer, all state changes occur before any potential external calls.

TC-08: TokenGroup Boundary Handling

Result: PASS

Description: The contract correctly handles token IDs at group boundaries (multiples of 256).

Evidence: This structure efficiently stores ownership data for up to 256 tokens in a single storage slot, with the burnBitmap tracking which tokens within the group have been burned.

Minting

```
Unset
function _mint(address to, uint256 quantity) internal virtual {
    // ...

    // Efficient storage of token groups
    uint256 updatedTokenId = startTokenId;
    for (uint256 i = 0; i < quantity; i += 256) {
        uint256 toMint = quantity - i;
        if (toMint > 256) {
            toMint = 256;
        }
        _owners[updatedTokenId / 256] = TokenGroup(to, 0);
        updatedTokenId += 256;
    }

    // ...
}
```

Source

Burn

```
Unset
function _isBurned(uint256 tokenId, uint256 burnBitmap) private pure returns (bool) {
    uint256 bit = tokenId % 256;
    return burnBitmap & (1 << bit) != 0;
}

...

function _burn(uint256 tokenId) internal virtual {
    // ...
```

```
        // Update burn bitmap in the token group
        uint256 tokenGroupIndex = tokenId / 256;
        TokenGroup memory tokenGroup = _owners[tokenGroupIndex];
        uint256 bit = tokenId % 256;
        tokenGroup.burnBitmap |= (1 << bit);
        _owners[tokenGroupIndex] = tokenGroup;

        // ...
    }
```

Source

ERC721PsiV2 Fuzzing Properties

ERC721PsiV2 is tested with Echidna. Echidna configuration can be found here.

Source code: contracts/test/token/erc721/fuzz/ERC721PsiV2.Echidna.sol at peter-erc721-perf-audit · immutable/contracts

Description	Tested	Passed
Total supply matches minted tokens	✓	✓
Balance consistency across all accounts	✓	✓
Balance sum matches total supply	✓	✓
All minted tokens have valid owners	✓	✓
Burned tokens have no owners	✓	✓
Non-existent tokens revert ownership checks	✓	✓
Group operations maintain consistency	✓	✓
Group occupancy limits respected	✓	✓
Group boundary operations work correctly	✓	✓
Cross-group operations maintain consistency	✓	✓
Minting sequence respects boundaries	✓	✓
Token ID boundary checks work	✓	✓
Quantity range validation works	✓	✓
Max token ID overflow protection	✓	✓