Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.09.10, the SlowMist security team received the Partyicons team's security audit application for Partyicons - NFT Assets, 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

This is the NFT Assets part of the Partyicons protocol, which mainly includes ERC721, ERC1155 and ERC2771

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risks of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	Missing null check	Others	Suggestion	Fixed

NO	Title	Category	Level	Status
N3	Missing event log	Others	Suggestion	Fixed
N4	Lack of checking whether tokenId exists	Design Logic Audit	Low	Fixed
N5	LockStatusChanged event not triggered	Design Logic Audit	Low	Fixed
N6	Token lock-up period is not set	Design Logic Audit	High	Fixed
N7	Missing whenNotPaused modifier	Design Logic Audit	Medium	Fixed
N8	The lockTime value is not checked	Others	Suggestion	Fixed
N9	Missing zero address check	Others	Suggestion	Fixed
N10	Incorrect setting of token lock time	Design Logic Audit	Medium	Fixed
N11	Missing zero address check	Others	Suggestion	Acknowledged
N12	Lack of checking whether tokenId exists	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/metaicons-lab/nft-assets

Initial audit version: 885c5b157287600047c5d2c0ce6af6b6e54f2c1c

Final audit version: e36ae38148072ef1fc7274dd629b9992cc7b66eb

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:

Box			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
safeTransferFrom	Public	Can Modify State	-
safeBatchTransferFrom	Public	Can Modify State	-
_checkTokenLock	Internal	-	-
addMinter	External	Can Modify State	onlyOwner
removeMinter	External	Can Modify State	onlyOwner
updateBaseURI	Public	Can Modify State	onlyOwner
uri	Public	-	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
setPrice	External	Can Modify State	onlyOwner
getPrice	External	-	-
setPaymentToken	External	Can Modify State	onlyOwner
mintBatch	Public	Can Modify State	whenNotPaused onlyMinter
mintAndLockBatch	Public	Can Modify State	whenNotPaused onlyMinter
openBox	External	Can Modify State	nonReentrant
isTokenLocked	Public	-	-

Box			
lockTokens	External	Can Modify State	onlyOwner
unlockTokens	External	Can Modify State	onlyOwner
_ownerOf	Internal	-	-
_tokenIdExists	Public	-	-
name	Public	-	-
symbol	Public	-	-
totalSupply	Public	-	-
_isAuthorizedSigner	Internal	-	-
mintWithSignature	External	Payable	-

BoxV2			
Function Name	Visibility	Mutability	Modifiers
mintWithSignature	External	Payable	-

Hero			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
totalSupply	Public	-	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

Hero			
burn	Public	Can Modify State	-
contractURI	Public	-	-
tokenURI	Public	-	-
updateBaseURI	External	Can Modify State	onlyOwner
_update	Internal	Can Modify State	-
_baseURI	Internal	-	-
supportsInterface	Public	-	-
setTrustedForwarder	External	Can Modify State	onlyOwner whenNotPaused
_msgSender	Internal	-	-
_msgData	Internal	-	-
_contextSuffixLength	Internal	-	-
setDefaultRoyalty	External	Can Modify State	onlyOwner
addMinter	External	Can Modify State	onlyOwner
removeMinter	External	Can Modify State	onlyOwner
version	External	-	-
_setTokenBaseURI	Internal	Can Modify State	-
mint	External	Can Modify State	whenNotPaused onlyMinter
mintBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant
burnBatch	External	Can Modify State	whenNotPaused nonReentrant
isTokenLocked	Public	-	-

Hero			
lockTokens	External	Can Modify State	-
unlockTokens	External	Can Modify State	-
mintAndLockBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant
_incrementTokenCounter	Internal	Can Modify State	-
_decrementTokenCounter	Internal	Can Modify State	-
_isAuthorizedSigner	Internal	-	-
mintWithSignature	External	Payable	-

Mirpass			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
totalSupply	Public	-	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
burn	Public	Can Modify State	-
contractURI	Public	-	-
tokenURI	Public	-	-
updateBaseURI	External	Can Modify State	onlyOwner

Mirpass			
_update	Internal	Can Modify State	-
_baseURI	Internal	-	-
supportsInterface	Public	-	-
setTrustedForwarder	External	Can Modify State	onlyOwner whenNotPaused
_msgSender	Internal	-	-
_msgData	Internal	-	-
_contextSuffixLength	Internal	-	-
setDefaultRoyalty	External	Can Modify State	onlyOwner
addMinter	External	Can Modify State	onlyOwner
removeMinter	External	Can Modify State	onlyOwner
version	External	-	-
_setTokenBaseURI	Internal	Can Modify State	-
mint	External	Can Modify State	whenNotPaused onlyMinter
mintBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant
burnBatch	External	Can Modify State	whenNotPaused nonReentrant
isTokenLocked	Public	-	-
lockTokens	External	Can Modify State	-
unlockTokens	External	Can Modify State	-
mintAndLockBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant

Mirpass			
_incrementTokenCounter	Internal	Can Modify State	-
_decrementTokenCounter	Internal	Can Modify State	-
_isAuthorizedSigner	Internal	-	-
mintWithSignature	External	Payable	-

Points			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
uri	Public	-	-
updateBaseURI	External	Can Modify State	onlyOwner
mintBatchForToken	Public	Can Modify State	onlyMinter
mintBatchForAccount	Public	Can Modify State	onlyMinter
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
addMinter	External	Can Modify State	onlyOwner
removeMinter	External	Can Modify State	onlyOwner
version	External	-	-
_isAuthorizedSigner	Internal	-	-
mintWithSignature	External	Payable	-

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

Weapon			
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyOwner
totalSupply	Public	-	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
burn	Public	Can Modify State	-
contractURI	Public	-	-
tokenURI	Public	-	-
updateBaseURI	External	Can Modify State	onlyOwner
_update	Internal	Can Modify State	-
_baseURI	Internal	-	-
supportsInterface	Public	-	-
setTrustedForwarder	External	Can Modify State	onlyOwner whenNotPaused
_msgSender	Internal	-	-
_msgData	Internal	-	-
_contextSuffixLength	Internal	-	-
setDefaultRoyalty	External	Can Modify State	onlyOwner
addMinter	External	Can Modify State	onlyOwner
removeMinter	External	Can Modify State	onlyOwner
version	External	-	-

Weapon			
_setTokenBaseURI	Internal	Can Modify State	-
mint	External	Can Modify State	whenNotPaused onlyMinter
mintBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant
burnBatch	External	Can Modify State	whenNotPaused nonReentrant
isTokenLocked	Public	-	-
lockTokens	External	Can Modify State	-
unlockTokens	External	Can Modify State	-
mintAndLockBatch	External	Can Modify State	whenNotPaused onlyMinter nonReentrant
_incrementTokenCounter	Internal	Can Modify State	-
_decrementTokenCounter	Internal	Can Modify State	-
_isAuthorizedSigner	Internal	-	-
mintWithSignature	External	Payable	-

4.3 Vulnerability Summary

[N1] [Medium] Risks of excessive privilege

Category: Authority Control Vulnerability Audit

Content

1.All contracts in this protocol use the UUPSUpgradeable upgrade mechanism from OpenZeppelin, which allows the `Owner` role to upgrade the smart contract.

- contracts/Box.sol #L8, L97-L99
- contracts/Hero.sol #L12, L77-L79

- contracts/Mirpass.sol #L12, L77-L79
- contracts/Points.sol #L9, L96-L98
- contracts/Weapon.sol #L12, L77-L79

```
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";

    function _authorizeUpgrade(
        address newImplementation
    ) internal override onlyOwner {}
```

2.In all contracts of this protocol, the `Owner` role can add or remove the `Minter` role through the `addMinter` function and `removeMinter` function.

- contracts/Box.sol #L130-L132, L134-L136
- contracts/Hero.sol #L225-L227, L233-L235
- contracts/Mirpass.sol #L224-L226, L232-L234
- contracts/Points.sol #L101-L103, L105-L107
- contracts/Weapon.sol #L224-L226, L232-L234

```
    function addMinter(address account) external onlyOwner {
        _updateMinter(account, true);
    }

    function removeMinter(address account) external onlyOwner {
        _updateMinter(account, false);
    }
```

3.In all contracts of this protocol, the `Owner` role can modify important parameters in the contract through the following functions.

- contracts/Box.sol #L138-L140 ,L173-L186, L199-L208

```
    function updateBaseURI
    function setPrice
    function setPaymentToken
```

- contracts/Hero.sol #L131-L133, L177-L181, L217-L222

- contracts/Mirpass.sol #L131-L133, L176-L180, L216-L221
- contracts/Weapon.sol #L131-L133, L176-L180, L216-L221

```
function updateBaseURI
function setTrustedForwarder
function setDefaultRoyalty
```

- contracts/Points.sol #L71-L73, L

```
function updateBaseURI
```

4.In all contracts of this protocol, the `Minter` role can mint tokens arbitrarily through the following functions.

- contracts/Box.sol #L216-L226, L228-L241, L334-L344

```
function mintBatch
function mintAndLockBatch
function mintWithSignature
```

- contracts/Hero.sol #L245-L250, L258-L269
- contracts/Mirpass.sol #L244-L249, L257-L268, L352-L369, L391-L404
- contracts/Weapon.sol #L244-L249, L257-L268, L352-L369, L391-L404

```
function mint
function mintBatch
function mintAndLockBatch
function mintWithSignature
```

- contracts/Points.sol #L75-L85, L87-L93, L122-L131

```
function mintBatchForToken
function mintBatchForAccount
function mintWithSignature
```

5.The `Owner` role in the Box contract, as well as the `Owner` and `Minter` roles in the Hero contract, Mirpass contract, and Weapon contract, can lock or unlock tokens through the `lockTokens` function and `unlockTokens` function.

- contracts/Box.sol #L277-L284, L290-L302
- contracts/Hero.sol #L296-L315, L321-L350
- contracts/Mirpass.sol #L295-L314, L320-L349
- contracts/Weapon.sol #L295-L314, L320-L349

```
function lockTokens
function unlockTokens
```

Solution

In the short term, transferring owner ownership 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

[N2] [Suggestion] Missing null check

Category: Others

Content

1.In the Box contract, the `initialize` function and the `updateBaseURI` function do not check that the value of `tokenURI_` parameter and `newuri` parameter is not null.

- contracts/Box.sol #L73-L95, L138-L140

```
function initialize(
    string calldata name_,
    string calldata symbol_,
    address paymentTokenAddress_,
    address minter_,
    string memory tokenURI_
) public initializer {
    __ERC1155_init(tokenURI_);
    ```
}
```
```

```
function updateBaseURI(string memory newuri) public onlyOwner {
    _setURI(newuri);
}
```

2.In the Hero, Mirpass, and Weapon contracts, the `_setTokenBaseURI` function did not check whether the `baseURI` parameter was not null.

- contracts/Hero.sol #L241-L243
- contracts/Mirpass.sol #L240-L242
- contracts/Weapon.sol #L240-L242

```
function _setTokenBaseURI(string memory baseURI_) internal {
    _tokenBaseURI = baseURI_;
}
```

3.In the Points contract, the `initialize` function and `updateBaseURI` function did not check whether the `tokenURI_` parameter and the `baseURI_` parameter were not null.

- contracts/Points.sol #L38-L53

```
function initialize(
    string calldata name_,
    string calldata symbol_,
    string memory tokenURI_,
    address minter_
) public initializer {
    __ERC1155_init(tokenURI_);
    ```
}

function updateBaseURI(string calldata baseURI_) external onlyOwner {
    setURI(baseURI);
}
```

**Solution**

It is recommended to check if the tokenURI_ parameter is not null.

**Status**

Fixed
```

[N3] [Suggestion] Missing event log

Category: Others

Content

1.In all contracts of the protocol, the `updateBaseURI` function lacks event logging when the base URI of the token is updated.

- contracts/Box.sol #L138-L140

```
function updateBaseURI(string memory newuri) public onlyOwner {
    _setURI(newuri);
}
```

- contracts/Hero.sol #L131-L133
- contracts/Mirpass.sol #L131-L133
- contracts/Weapon.sol #L131-L133

```
function updateBaseURI(string calldata baseURI_) external onlyOwner {
    _setTokenBaseURI(baseURI_);
}
```

- contracts/Points.sol #L71-L73

```
function updateBaseURI(string calldata baseURI_) external onlyOwner {
    _setURI(baseURI_);
}
```

2.In all contracts of this protocol, the `ISignatureMintERC721` interface defines the `TokensMintedWithSignature` event, which is used to record token minting with signatures. However, this event is missing in the `mintWithSignature` function.

- contracts/Box.sol #L334-L344

```
function mintWithSignature(
    MintRequest calldata _req,
    bytes calldata _signature
) external virtual payable returns (address signer) {
    signer = _processRequest(_req, _signature);
```

```
        address receiver = _req.to;
        uint256 tokenId = _req.tokenId;
        uint256 quantity = _req.quantity;
        _mint(receiver, tokenId, quantity, "");
        _totalSupply[tokenId] += quantity;
    }
```

- contracts/Hero.sol #L392-L405
- contracts/Mirpass.sol #L391-L404
- contracts/Weapon.sol #L390-L403

```
    function mintWithSignature(
        MintRequest calldata _req,
        bytes calldata _signature
    ) external virtual payable returns (address signer) {
        if (_req.quantity != 1) {
            revert SignatureMintInvalidQuantity();
        }

        signer = _processRequest(_req, _signature);

        address receiver = _req.to;
        uint256 tokenId = _req.tokenId;
        _safeMint(receiver, tokenId);
    }
```

- contracts/Points.sol #L122-L131

```
    function mintWithSignature(
        MintRequest calldata _req,
        bytes calldata _signature
    ) external virtual payable returns (address signer) {
        signer = _processRequest(_req, _signature);
        address receiver = _req.to;
        uint256 tokenId = _req.tokenId;
        uint256 quantity = _req.quantity;
        _mint(receiver, tokenId, quantity, "");
    }
```

Solution

It is recommended to add event logging.

Status

Fixed

[N4] [Low] Lack of checking whether tokenId exists

Category: Design Logic Audit

Content

1.In the BOX contract, the `uri` function, `setPrice` function, `isTokenLocked` function, `lockTokens` function, and `unlockTokens` function do not check whether the `tokenId` parameter exists.

- contracts/Box.sol#L142-L152, L173-L186, L267-L271, L277-L284, L290-L302

```
    function uri(uint256 tokenId) public virtual view override returns (string
memory) {
        ```
    }

    function setPrice(
        uint256 tokenId_,
        uint256 pointTokenId_,
        uint256 salePrice_,
        address newPaymentTokenAddress_
) external virtual onlyOwner {
        ```
    }

    function isTokenLocked(
        uint256 tokenId
    ) public virtual view override returns (bool) {
        return _lockedTokens[tokenId];
    }

    function lockTokens(
        uint256[] calldata tokenIds
    ) external virtual override onlyOwner {
        ```
    }

    function unlockTokens(
        uint256[] calldata tokenIds
) external virtual override onlyOwner {
        ```
    }
```

2.In the Points contract, the `uri` function does not check whether the `tokenId` parameter exists.

- contracts/Points.sol #L55-L65

```
    function uri(uint256 tokenId) public virtual view override returns (string
memory) {
        ```
    }
```

3.In the Hero contract, the Mirpass contract and the Weapon contract, the `tokenURI` function, `isTokenLocked` function, `lockTokens` function, and `unlockTokens` function do not check whether the `tokenId` parameter exists.

- contracts/Hero.sol #L286-L290, L296-L315, L321-L350
- contracts/Mirpass.sol #L285-L289, L295-L314, L320-L349
- contracts/Weapon.sol #L285-L289, L295-L314, L320-L349

```
    function isTokenLocked(
        uint256 tokenId
) public virtual view override returns (bool) {
        return _lockedTokens[tokenId];
    }

    function lockTokens(uint256[] calldata tokenIds) external virtual override {
        ```
    }

    function unlockTokens(uint256[] calldata tokenIds) external virtual override {
        ```
    }
```

**Solution**

It is recommended that the function first check whether tokenId exists before continuing to operate on it.

**Status**

Fixed

**[N5] [Low] LockStatusChanged event not triggered**
```

Category: Design Logic Audit

Content

In the BOX contract,the `mintAndLockBatch` function locks tokens while minting them in batches, but the `LockStatusChanged` event is not triggered in the function to record the operation of locking tokens.

- contracts/Box.sol#L228-L241

```
    function mintAndLockBatch(
        uint256 tokenId,
        address[] calldata to,
        uint256[] calldata amount,
        uint256 lockTime
    ) public virtual whenNotPaused onlyMinter {
        require(to.length == amount.length, "Invaild input");
        for (uint256 i = 0; i < to.length; i++) {
            _mint(to[i], tokenId, amount[i], "");
            _totalSupply[tokenId] += amount[i];
            _lockedTokens[tokenId] = true;
            _lockTime[tokenId] = lockTime;
        }
    }
```

Solution

It is recommended to add the LockStatusChanged event to record the operation of locking tokens.

Status

Fixed

[N6] [High] Token lock-up period is not set

Category: Design Logic Audit

Content

In the Box contract, Hero contract, Mirpass contract, and Weapon contract, the `lockTokens` function is used to lock tokens, but the lock time is not set. In particular, in the Hero contract, Mirpass contract, and Weapon contract, if the `Minter` role locks a token with a `tokenId`, since the lock time is not set, the owner of the token can directly unlock the token through the `unlockTokens` function.

- contracts/Box.sol #L277-L284

```
function lockTokens(
    uint256[] calldata tokenIds
) external virtual override onlyOwner {
    for (uint i = 0; i < tokenIds.length; i++) {
        _lockedTokens[tokenIds[i]] = true;
        emit LockStatusChanged(tokenIds[i], true, _msgSender());
    }
}
```

- contracts/Hero.sol #L296-L315
- contracts/Mirpass.sol #L295-L314
- contracts/Weapon.sol #L295-L314

```
function lockTokens(uint256[] calldata tokenIds) external virtual override {
    address sender = _msgSender();
    if (_isMinter(sender)) {
        for (uint i = 0; i < tokenIds.length; i++) {
            _lockedTokens[tokenIds[i]] = true;
            emit LockStatusChanged(tokenIds[i], true, sender);
        }
    } else {
        for (uint i = 0; i < tokenIds.length; i++) {
            if (_msgSender() == ownerOf(tokenIds[i])) {
                _lockedTokens[tokenIds[i]] = true;
                emit LockStatusChanged(tokenIds[i], true, sender);
            } else {
                revert(
                    "Only the owner or the minter can modify the lock status"
                );
            }
        }
    }
}
```

Solution

It is recommended to set the lock-up period of tokens in the function.

Status

Fixed

[N7] [Medium] Missing whenNotPaused modifier

Category: Design Logic Audit

Content

In the Box contract, Hero contract, Mirpass contract, and Weapon contract, the PausableUpgradeable module of openzeppelin is used to lock the token minting function of the contract. However, the `mintWithSignature` function does not add the `whenNotPaused` modifier, which means that when the contract is in a locked state, the `Minter` role can mint tokens through the `mintWithSignature` function, thereby bypassing the contract lock state.

- contracts/Box.sol #L334-L344

```
    function mintWithSignature(
        MintRequest calldata _req,
        bytes calldata _signature
    ) external virtual payable returns (address signer) {
        signer = _processRequest(_req, _signature);
        address receiver = _req.to;
        uint256 tokenId = _req.tokenId;
        uint256 quantity = _req.quantity;
        _mint(receiver, tokenId, quantity, "");
        _totalSupply[tokenId] += quantity;
    }
```

- contracts/Hero.sol #L392-L405
- contracts/Mirpass.sol #L391-L404
- contracts/Weapon.sol #L390-L403

```
    function mintWithSignature(
        MintRequest calldata _req,
        bytes calldata _signature
    ) external virtual payable returns (address signer) {
        if (_req.quantity != 1) {
            revert SignatureMintInvalidQuantity();
        }

        signer = _processRequest(_req, _signature);

        address receiver = _req.to;
        uint256 tokenId = _req.tokenId;
        _safeMint(receiver, tokenId);
    }
```

Solution

It is recommended to add a whenNotPaused modifier to the mintWithSignature function.

Status

Fixed

[N8] [Suggestion] The lockTime value is not checked

Category: Others

Content

In the Hero contract, Mirpass contract and Weapon contract, the `mintAndLockBatch` function did not check if the `lockTime` parameter was greater than 0.

- contracts/Hero.sol #L353-L370
- contracts/Mirpass.sol #L352-L369
- contracts/Weapon.sol #L352-L369

```
    function mintAndLockBatch(
        address[] calldata addressList,
        uint256[] calldata tokenIds,
        uint256 lockTime
    ) external virtual whenNotPaused onlyMinter nonReentrant {
        require(
            addressList.length > 0 && addressList.length == tokenIds.length,
            "Invalid input"
        );
        address sender = _msgSender();
        for (uint i = 0; i < addressList.length; i++) {
            uint256 tokenId = tokenIds[i];
            _safeMint(addressList[i], tokenId);
            _lockedTokens[tokenId] = true;
            _lockTime[tokenId] = block.timestamp + lockTime;
            emit LockStatusChanged(tokenId, true, sender);
        }
    }
```

Solution

It is recommended to check whether the lockTime parameter is greater than 0 in the function.

Status

Fixed

[N9] [Suggestion] Missing zero address check

Category: Others

Content

In the Box contract, the `setPaymentToken` function did not perform a zero address check on the `newPaymentTokenAddress_` parameter.

- contracts/Box.sol #L199-L208

```
    function setPaymentToken(
        address newPaymentTokenAddress_
    ) external virtual onlyOwner {
        address oldPaymentToken = defaultPaymentAddress;
        defaultPaymentAddress = newPaymentTokenAddress_;
        emit DefaultPaymentAddressChanged(
            newPaymentTokenAddress_,
            oldPaymentToken
        );
    }
```

Solution

It is recommended to add a zero address check.

Status

Fixed

[N10] [Medium] Incorrect setting of token lock time

Category: Design Logic Audit

Content

In the Box contract, the `mintAndLockBatch` function incorrectly sets the lock time to `lockTime` when setting the token lock, causing the lock time to deviate from the expected value.

- contracts/Box.sol #L228-L241

```
    function mintAndLockBatch(
        uint256 tokenId,
        address[] calldata to,
        uint256[] calldata amount,
        uint256 lockTime
    ) public virtual whenNotPaused onlyMinter {
        require(to.length == amount.length, "Invaild input");
        for (uint256 i = 0; i < to.length; i++) {
            _mint(to[i], tokenId, amount[i], "");
            _totalSupply[tokenId] += amount[i];
            _lockedTokens[tokenId] = true;
            _lockTime[tokenId] = lockTime;
        }
    }
```

Solution

It is recommended to set the lock time to block.timestamp + lockTime.

Status

Fixed

[N11] [Suggestion] Missing zero address check

Category: Others

Content

In the Hero contract, Mirpass contract, and Weapon contract, the `initialize` function and the `setTrustedForwarder` function do not perform a zero address check on the `trustedForwarder_` parameter.

- contracts/Hero.sol #L55-L71, L177-L181
- contracts/Mirpass.sol #L55-L71, L176-L180
- contracts/Weapon.sol #L55-L71, L176-L180

```
    function initialize(
        string calldata name_,
        string calldata symbol_,
        string calldata baseURI_,
        address minter_,
        address trustedForwarder_
    ) public initializer {
        ```
        __ERC2771_init(trustedForwarder_);
```
```

```
        ```
    }

    function setTrustedForwarder(
        address trustedForwarder_
) external onlyOwner whenNotPaused {
        setTrustedForwarder(trustedForwarder);
    }
```

**Solution**

It is recommended to add a zero address check.

**Status**

Acknowledged

### [N12] [Suggestion] Lack of checking whether tokenId exists

**Category: Design Logic Audit**

**Content**

In the Hero contract, the Mirpass contract and the Weapon contract, the `tokenURI` function does not check whether the `tokenId` parameter exists.

- contracts/Hero.sol #L120-L125
- contracts/Mirpass.sol #L120-L125
- contracts/Weapon.sol #L120-L125

```
    function tokenURI(
        uint256 tokenId_
    ) public view override(ERC721Upgradeable) returns (string memory) {
        string memory URI = super.tokenURI(tokenId_);
        return string(abi.encodePacked(URI, ".json"));
    }
```

Solution

It is recommended that the function first check whether tokenId exists before continuing to operate on it.

Status

Acknowledged
```

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002409130001	SlowMist Security Team	2024.09.10 - 2024.09.13	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 3 medium risk, 2 low risk, 6 suggestion.

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.