HAECHI AUDIT

Intella X LaunchPad

Smart Contract Security Analysis

Published on : Oct 13, 2022

Version v1.1

HAECHI AUDIT

Smart Contract Audit Certificate

Intella X LaunchPad

Security Report Published by HAECHI AUDIT

v1.1 Oct 13, 2022

Auditor : Andy Koo, Paul Kim

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	1	1	-	-
Medium	2	2	-	-
Low	1	1	-	-
Tips	1	1	-	-

TABLE OF CONTENTS

5 Issues (0 Critical, 1 High, 2 Medium, 1 low, 1 Tips) Found, all issues were resolved.

TABLE OF CONTENTS

ABOUT US

Executive Summary

OVERVIEW

- Protocol overview
- Scope
- Access Controls

FINDINGS

- Minted Token URI can be duplicated when a user burns the NFT on the LaunchPad.
- The allocated token for the project team should be minted before open the minting round.
- The check-effect-interaction pattern on the minting process is imperfect.
- Migration of NFT from launchpad should be conducted after the reveal is confirmed.
- Several important variable changes have no event emissions.

DISCLAIMER

Appendix. A

- Severity Level
- Difficulty Level
- Vulnerability Category

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 400+ 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

Executive Summary

Purpose of this report

This report was prepared to audit the security of the NFT Launchpad contracts developed by the Intella X team. HAECHI AUDIT conducted the audit focusing on whether the system created by the Intella X team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the NFT Launchpad implementation.
In detail, we have focused on the following

- Unintended behavior on the NFT minting process.
- Project availability issues like Denial of Service.
- Storage variable access control.
- Possibility of exploiting and abusing predictability of the seed number.
- Adequate implementation of ERC721 spec and ChainLink VRF.
- Existence of known smart contract vulnerabilities

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/tree/main/contracts/launchpad).

The last commit of the code used for this Audit is "611dd35bdf750245f56e7a1f1555a235dc19f899".

Audit Timeline

Date	Event
2022/09/13	Audit Initiation (NFT Launchpad)
2022/09/27	Delivery of v1.0 report.
2022/10/07	Fix commit pushed to repository.
2022/10/13	Delivery of v1.1 report.

Findings

HAECHI AUDIT found 1 High, 2 medium and 1 Low severity issues. There are 1 Tips issues explained that would improve the code's usability or efficiency upon modification. We have verified that all issues have been fixed.

Severity	Issue	Status
High	Minted Token URI can be duplicated when a user burns the NFT on the LaunchPad.	(Resolved - v1.1)
Medium	The allocated token for the project team should be minted before open the minting round.	(Resolved - v1.1)
Medium	The check-effect-interaction pattern on the minting process is imperfect.	(Resolved - v1.1)
Low	Migration of NFT from launchpad should be conducted after the reveal is confirmed.	(Resolved - v1.1)
TIPS	Several important variable changes have no event emissions.	(Resolved - v1.1)

Code Maturity

Criteria	Status	Comment
Level of Documentation	Medium	Natspec comments exist on the codebase. However, some functions are remained undocumented.
Test Coverage	High	Unit Tests are well organized and coverage is sufficient.

Remarks

The project uses ERC-2771 meta transaction and the trusted forwarder has the ability to forward the user's contract call. The forwarder is implemented using the OpenGSN protocol and this implementation is not included in the audit scope.

OVERVIEW

Protocol overview

• LaunchPad

The project team can add rounds to the LaunchPad contract. The round information consists of the maximum minting amount and maximum minting amount per user, and the time scope of the round and minting price. The minting round can be public or private. In the case of a private minting round, a user should be able to prove that their address is included in the Merkle root provided by the project team.

The project uses ChainLink's verifiable random function(VRF) to generate the seed number of tokenID. After the random seed is revealed, the URI of the NFT token can be observed.

The minted NFT implements the ERC-2981 royalty standard and ERC-4494 tokenID-nonce based permit. The contract owner can set the royalty information on the specific tokenID and the permit nonce is increased by 1 after token transfer.

• NFT Migration

The user can migrate the NFT that is minted on the LaunchPad to the NeowizGameItem contract. The NFT migration is only available by using the allowed LaunchPad address. The migration burns the NFT on the LaunchPad contract and mints a new NFT on the NeowizGameItem contract using the tokenURI after the random seed number is set by VRF.

The contract is deployed using the UUPS proxy contract.

Scope

contracts

```
├── BlockhashNeowizERC721.sol
├── ChainLinkNeowizERC721.sol
├── DataProvider.sol
├── NeowizERC721.sol
├── NeowizGameItem.sol
├── lib
│   └── metatx
│       └── ERC2771ContextFromStorage.sol
├── external
│   └── erc721-with-permits
│       ├── ERC721WithPermit.sol
│       ├── ERC721WithPermitUpgradeable.sol
│       └── IERC721WithPermit.sol
└── interfaces
    ├── IERC20Permit.sol
    └── INeowizERC721.so
```

Access Controls

NFT Launchpad contracts have the following access control mechanisms.

- onlyOwner()
- onlyAllowed()

onlyOwner() : modifier that controls access to variables including launchpad address, and settings related to NFT minting round, trusted forwarder address for ERC-2771.

- BlockhashNeowizERC721#requestRandomSeed()
- ChainLinkNeowizERC721#requestRandomSeed()
- NeowizERC721#setPayment()
- NeowizERC721#setUnRevealedURI()
- NeowizERC721#addRound()
- NeowizERC721#updateState()
- NeowizERC721#updateMaxMint()
- NeowizERC721#updateMaxMintPerAccount()
- NeowizERC721#updatePrice()
- NeowizERC721#updateMerkleRoot()
- NeowizERC721#updateRoundTimestamp()
- NeowizERC721#setBaseURI()
- NeowizERC721#setRoyaltyInfo()
- NeowizERC721#mintResidue()
- NeowizERC721#teamMint()
- NeowizERC721#withdraw()
- NeowizGameItem#mint()
- NeowizGameItem#allowLaunchpad()
- NeowizGameItem#denyLaunchpad()

onlyAllowed() : modifier that controls migrate function's allowed LaunchPad address check.

- NeowizGameItem#migrate()

The owner has permissions that can change the crucial part of the system. It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes.

FINDINGS

Minted Token URI can be duplicated when a user burns the NFT on the LaunchPad.

ID: INTELLAX-01
Type: Logic Error
File: contracts/NeowizERC721.sol

Severity: HIGH
Difficulty: Low

Issue

The *NeowizERC721.tokenURI()* function returns a distinct token URI per token ID. The 'distinct' means the returned token URI should not be duplicated. However, because the *burn()* function is available, this condition can be broken.

```
/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
/// @dev Throws if `_tokenId` is not a valid NFT.
function tokenURI(uint256 tokenId)
    public
    view
    virtual
    override
    returns (string memory)
{
    _requireMinted(tokenId);

    if (!revealed) return unrevealedURI;

    uint256 shiftedId;
    if (tokenId < TEAM_SUPPLY) {
        shiftedId = tokenId;
    } else {
        shiftedId =
            ((tokenId - TEAM_SUPPLY + randomSeed) %
                (MAX_TOTAL_SUPPLY - TEAM_SUPPLY)) +
            TEAM_SUPPLY;
    }

    return super.tokenURI(shiftedId);
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L509]

The *shiftedID* is calculated based on the token ID which is set when the NFT is minted. This token ID is monotonically increased by 1. If a user intentionally or inadvertently burns the minted NFT, the token ID can not be reused and would decrease the return value of *totalSupply()*.

```
/**
 * @dev Returns the total number of tokens in existence.
 * Burned tokens will reduce the count.
 * To get the total number of tokens minted, please see {_totalMinted}.
 TEAM_SUPPLY = _currentIndex = _teamSupply;
 */
function totalSupply() public view returns (uint256) {
    // Counter underflow is impossible as _burnCounter cannot be incremented more than
`_currentIndex` times.
    unchecked {
        return totalMinted() - _burnCounter;
    }
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L466]

Consequently, the token ID can exceed the expected value within the MAX_TOTAL_SUPPLY. As the *shiftedId* is the modulo of *(MAX_TOTAL_SUPPLY - TEAM_SUPPLY)*, the minted token ID after burn would be calculated to duplicate *shiftedId*.

```
it.only("tokenURI() - Non duplication check", async function () {
    let currentTimestamp = BigNumber.from(await getBlockTimestamp());
    let currentBlockNumber = await getBlockNumber();
    await chainLinkNeowizERC721.addRound(
        SaleState.PUBLIC,
        300,
        1000,
        ethers.utils.parseEther("1"),
        merkleRoot,
        currentTimestamp.add(100),
        currentTimestamp.add(10000),
    );

    mineBlocks(10);
    mineTime(101);
    await chainLinkNeowizERC721.teamMint(100);
    await chainLinkNeowizERC721.connect(user1).publicMint(300, 1, { value:
ethers.utils.parseEther("300") });
    await chainLinkNeowizERC721.connect(user2).publicMint(300, 1, { value:
ethers.utils.parseEther("300") });
    await chainLinkNeowizERC721.connect(user3).publicMint(300, 1, { value:
ethers.utils.parseEther("300") });
    await chainLinkNeowizERC721.connect(user3).burn(999);
    await chainLinkNeowizERC721.connect(user3).burn(998);
    await chainLinkNeowizERC721.connect(user4).publicMint(2, 1, { value:
ethers.utils.parseEther("2") });
    await chainLinkNeowizERC721.connect(coordinator).rawFulfillRandomWords(901, [901], { from:
coordinator.address });

    expect(await chainLinkNeowizERC721.tokenURI(1000)).not.to.be.equal(await
chainLinkNeowizERC721.tokenURI(100));
    expect(await chainLinkNeowizERC721.tokenURI(1001)).not.to.be.equal(await
chainLinkNeowizERC721.tokenURI(101));
});
```

[Duplication unit test code]

The test code shows that 2 newly minted tokens after burns have the same tokenURI as other token IDs.

```
tokenID 1000 = http://hello.world/101
tokenID 1001 = http://hello.world/102
tokenID 100 = http://hello.world/101
tokenID 101 = http://hello.world/102
```

[Duplication unit test result]

Recommendation

The *burn()* function on the LaunchPad contract can be only callable by the *NeowizGameItem* contract. In other ways, The *burn()* function can be restricted until the public mint is finished.

Update

The Fix[1470adce] modified *_mintRound()* function from comparing minted quantity with *MAX_TOTAL_SUPPLY* to comparing residual mintable amount. That means, a user cannot mint the NFT that has token ID out of allowed range.

```
      function _mintInRound(uint256 _round, address _to, uint256 _quantity) internal {
          ...
---       if (MAX_TOTAL_SUPPLY < totalSupply() + _quantity)
---           revert MaxTotalSupplyExceeded();

+++       if (_notTeamResidue() < _quantity) {
+++           revert MaxTotalSupplyExceeded();
+++       }
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/pull/2/commits/1470adcee9d820df248bd4951825d3d165d66f32]

The allocated token for the project team should be minted before open the minting round.

ID: INTELLAX-02
Type: Logic Error
File: contracts/NeowizERC721.sol

Severity: Medium
Difficulty: Medium

Issue

The mint amount is capped to *MAX_TOTAL_SUPPLY* which is set on the contract creation. This amount is checked on the mint process with the return value of the totalSupply() function.

```
/**
 * @dev Returns the total number of tokens in existence.
 * Burned tokens will reduce the count.
 * To get the total number of tokens minted, please see {_totalMinted}.
 TEAM_SUPPLY = _currentIndex = _teamSupply;
 */
function totalSupply() public view returns (uint256) {
    // Counter underflow is impossible as _burnCounter cannot be incremented more than
`_currentIndex` times.
    unchecked {
        return totalMinted() - _burnCounter;
    }
}

/**
 * @notice Returns the total amount of tokens minted in the contract.
 */
function totalMinted() public view returns (uint256) {
    unchecked {
        return _currentIndex - TEAM_SUPPLY + _currentTeamIndex;
    }
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L466]

The *MAX_TOTAL_SUPPLY* includes the *TEAM_SUPPLY* such that *totalMinted()* considers the *_currentTeamIndex* value. The *_currentTeamIndex* value is increased to *TEAM_SUPPLY* when the actual allocated NFT token is minted.

```
function _mintTeamQuantity(address _to, uint256 _quantity) private {
    for (
        uint256 i = _currentTeamIndex;
        i < _currentTeamIndex + _quantity;
        i++
    ) {
        _safeMint(_to, i);
    }
    _currentTeamIndex += _quantity;
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L430]

In case the project team does not mint the allocated NFT tokens, the *_currentTeamIndex* value would remain 0 and this means the user can mint an excessive amount of tokens.

For example, let *MAX_TOTAL_SUPPLY*=1000 , *TEAM_SUPPLY=100*, if the project team didn't minted the 100 NFT tokens before start public minting, users can mint 1000 tokens. As token ID below 100 is allocated exclusively to the project team, 100 duplicated token URIs would be generated.

Recommendation

To prevent excessive minting, *_currentTeamIndex* should be checked before the minting round is added.

Update

The Fix[1470adce] modified *_mintRound()* function from comparing minted quantity with *MAX_TOTAL_SUPPLY* to comparing residual mintable amount. That means, a user cannot mint the NFT that has token ID out of allowed range.

```
      function _mintInRound(uint256 _round, address _to, uint256 _quantity) internal {
          ...
---       if (MAX_TOTAL_SUPPLY < totalSupply() + _quantity)
---           revert MaxTotalSupplyExceeded();

+++       if (_notTeamResidue() < _quantity) {
+++           revert MaxTotalSupplyExceeded();
+++       }
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/pull/2/commits/1470adcee9d820df248bd4951825d3d165d66f32]

The check-effect-interaction pattern on the minting process is imperfect.

ID: INTELLAX-03	Severity: Medium
Type: Reentrancy	Difficulty: High
File: contracts/NeowizERC721.sol	

Issue

The *_mintInRound()* function has 2 external calls. The first is *payment.transferFrom()* and this interaction follows the Chek-Effect-Interaction pattern. The second is the on*ERC721Received()* function which is called inside *_mintNotTeamQuantity()* function.

```
function _mintInRound(
    uint256 _round,
    address _to,
    uint256 _quantity
) internal {
    // check
    Round storage round = rounds[_round];
    uint256 totalPrice = round.price * _quantity;
    if (address(payment) != address(0)) {
        if (payment.balanceOf(_to) < totalPrice)
            revert NotEnoughERC20Fund();
    } else {
        if (msg.value < totalPrice) revert NotEnoughFund();
    }
    if (round.maxMintPerAccount < round.numberMinted[_to] + _quantity)
        revert MaxMintPerAccountExceeded(_round, _to);

    // Skip round.maxMint check if it is the last round
    if (
        _round != numRounds &&
        round.maxMint < (round.totalMinted + _quantity)
    ) revert MaxMintExceeded(_round);
    if (MAX_TOTAL_SUPPLY < totalSupply() + _quantity)
        revert MaxTotalSupplyExceeded();

    // effect
    round.numberMinted[_msgSender()] += _quantity;
    round.totalMinted += _quantity;
    _mintNotTeamQuantity(_to, _quantity);

    // interaction
    if (address(payment) != address(0)) {
        payment.transferFrom(_to, address(this), totalPrice);
    }
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L381]

The _safeMint() calls the on*ERC721Received()* function to the address of *_to*. For the completeness of the Chek-Effect-Interaction pattern, *_currentIndex* should be updated before the *_safeMint()* loop.

```
function _mintNotTeamQuantity(address _to, uint256 _quantity) private {
    for (uint256 i = _currentIndex; i < _currentIndex + _quantity; i++) {
        _safeMint(_to, i);
    }
    _currentIndex += _quantity;
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L442]

```
function onERC721Received(
    address,
    address,
    uint256 _tokenId,
    bytes calldata
) external payable returns (bytes4) {
    if (aggregated_burn < 3) {
        aggregated_burn++;
        malicious_burn(_tokenId);
        return IERC721Receiver.onERC721Received.selector;
    } else {
        return IERC721Receiver.onERC721Received.selector;
    }
}
```

[Example code of malicious onERC721Received]

Recommendation

To prevent unintended behavior using *onERC721Received()* function, all the effective state updates should be implemented before interaction.

Update

The *_mintRount()* function updates *_currentIndex* before calling the mint function and follows a check-effect-interaction pattern. Fix[1470adce].

```
    function _mintInRound(uint256 _round, address _to, uint256 _quantity) internal {
        ...
      // effect
      round.numberMinted[_msgSender()] += _quantity;
      round.totalMinted += _quantity;
      _currentIndex += _quantity;

      // interaction
      _mintNotTeamQuantity(_to, _quantity);
        ...
     }
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/pull/2/commits/1470adcee9d820df248bd4951825d3d165d66f32]

Migration of NFT from launchpad should be conducted after the reveal is confirmed.

ID: INTELLAX-04	Severity: Low
Type: Logic Error	Difficulty: Low
File: contracts/NeowizGameItem.sol	

Issue

The *NeowizGameItem.migrate()* function mint new NFT using the user's NFT of LaunchPad contract and burns the Launchpad NFT. A user can accidentally approve the LaunchPad NFT to *NeowizGameItem* contract before revealing the token URI.

```
function migrate(address _launchpad, uint256 _tokenId)
    external
    override
    onlyAllowed(_launchpad)
{
    string memory uri = IERC721Metadata(_launchpad).tokenURI(_tokenId);
    address tokenOwner = IERC721(_launchpad).ownerOf(_tokenId);
    _mintWithURI(tokenOwner, uri);

    IERC721Metadata(_launchpad).transferFrom(
        tokenOwner,
        address(this),
        _tokenId
    );
    INeowizERC721(_launchpad).burn(_tokenId);
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizGameItem.sol#L109]

The token URI before reveal is set to default URI, which means calling the *migrate()* function before reveal makes the purchased NFT useless.

```
function tokenURI(uint256 tokenId)
    public
    view
    virtual
    override
    returns (string memory)
{
    _requireMinted(tokenId);

    if (!revealed) return unrevealedURI;

    ...
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/blob/main/contracts/NeowizERC721.sol#L509]

Recommendation

To check whether the reveal was conducted on the *migrate()* function can prevent accidental burning of purchased NFT.

Update

The *migrate()* function modified to check the reveal status of the launchpad contract.

Fix[1470adce].

```
function migrate(address _launchpad, uint256 _tokenId) external override onlyAllowed(_launchpad) {
    string memory uri = IERC721Metadata(_launchpad).tokenURI(_tokenId);
    address tokenOwner = IERC721(_launchpad).ownerOf(_tokenId);
    _mintWithURI(tokenOwner, uri);

    if (!INeowizERC721(_launchpad).isRevealed()) {
        revert NotRevealed();
    }

    IERC721Metadata(_launchpad).transferFrom(tokenOwner, address(this), _tokenId);
    INeowizERC721(_launchpad).burn(_tokenId);
}
```

[https://github.com/ModoriLabs/neowiz-nft-marketplace-hardhat/pull/2/commits/1470adcee9d820df248bd4951825d3d165d66f32]

Several important variable changes have no event emissions.

ID: INTELLAX-05	Severity: N/A
Type: Tips	Difficulty: N/A
File: contracts/*	

Issue

Several important variable changes have no event emissions.

- `NeowizGameItem#allowLaunchpad()`
- `NeowizGameItem#denyLaunchpad()`
- `NeowizERC721#setTrustedForwarder()`
- `NeowizERC721#setPayment()`
- `NeowizERC721#setUnRevealedURI()`
- `NeowizERC721#addRound()`
- `NeowizERC721#updateState()`
- `NeowizERC721#updateMaxMint()`
- `NeowizERC721#updateMaxMintPerAccount()`
- `NeowizERC721#updatePrice()`
- `NeowizERC721#updateMerkleRoot()`
- `NeowizERC721#updateRoundTimestamp()`
- `NeowizERC721#setBaseURI()`
- `NeowizERC721#setRoyaltyInfo()`

[Functions without event emission]

Recommendation

Emitting events on the critical variable changes can enhance the visibility of the project.

Update

After Fix[1470adce], the changes of storage variables emit events.

DISCLAIMER

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

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	• Proper use of Check-Effect-Interact pattern. • Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	• Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document