NFT Launchpad

Smart Contract Audit Report
Prepared for Ancient8

Date Issued:	Feb 7, 2023
Project ID:	AUDIT2023001
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2023001
Version	v1.0
Client	Ancient8
Project	NFT Launchpad
Auditor(s)	Wachirawit Kanpanluk Kongkit Chatchawanhirun
Author(s)	Wachirawit Kanpanluk Kongkit Chatchawanhirun
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Feb 7, 2023	Full report	Wachirawit Kanpanluk Kongkit Chatchawanhirun

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary **1**
1.1. Audit Result 1
1.2. Disclaimer 1

2. Project Overview **2**
2.1. Project Introduction 2
2.2. Scope 3

3. Methodology **4**
3.1. Test Categories 4
3.2. Audit Items 5
3.3. Risk Rating 7

4. Summary of Findings **8**

5. Detailed Findings Information **10**
5.1. Missing Access Control 10
5.2. Improper msg.value Validation 15
5.3. Design Flaw in buyNft() Function 18
5.4. Smart Contract with Unpublished Source Code 25
5.5. Unbound Configuration Parameter in the NFT contract 26
5.6. Inexplicit Solidity Compiler Version 30
5.7. Insufficient Logging for Privileged Functions 31
5.8. Improper Function Visibility 33

6. Appendix **35**
6.1. About Inspex 35

1. Executive Summary

As requested by Ancient8, Inspex team conducted an audit to verify the security posture of the NFT Launchpad smart contracts on Jan 10, 2023. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of NFT Launchpad smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 critical, 1 medium, 2 low, 1 very low, and 3 info-severity issues. With the project team's prompt response, 1 critical, 1 medium, 1 very low, and 2 info-severity issues were resolved or mitigated in the reassessment, while 2 low and 1 info-severity issues were acknowledged by the team. Therefore, Inspex trusts that NFT Launchpad smart contracts have sufficient protections to be safe for public use. However, as the source code is not publicly available, the bytecode of the smart contracts deployed should be compared with the bytecode of the smart contracts audited before interacting with them. In the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Ancient8 - NFTLaunchpad is a contract that can be used to create an NFT box, which is the NFT contract. It allows the platform's users to participate in buying the NFT.

The NFT box has three types of sale times: private sale, public sale, and combination sale. The private and combination sale times have the whitelist who has a chance to buy before the public users. Public users are allowed to buy after that.

All of the NFTs that have been bought will be delivered in the same order as the order of NFTs that was created by the NFT contract, so this is like a "first come, first served" manner.

Scope Information:

Project Name	NFT Launchpad
Website	https://ancient8.gg
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	NFT, Launchpad

Audit Information:

Audit Method	Whitebox
Audit Date	Jan 10, 2023
Reassessment Date	Jan 16, 2023

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The smart contracts with the following bytecodes were audited and reassessed by Inspex in detail:

Initial Audit:

Contract	Bytecode SHA256 Hash
NFT	f962399c3e24fc2b75251e00742165f6325ad6cd48f7d9389f906edf001eeeec
NFTLaunchpad	f73686987c5d5f148b3abb4c278a131612727a4a8d5fa8237f86f9403e62e970

Reassessment Audit:

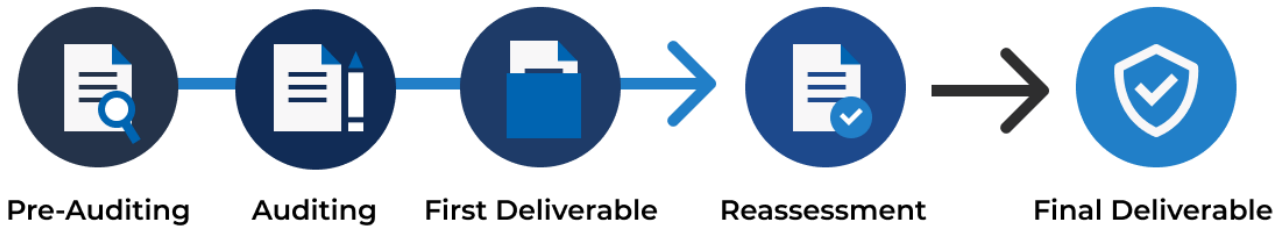
Contract	Bytecode SHA256 Hash
NFT	791d4362ede50f5c1e10ed83bf4d2fad222332965c077f84c55e47c6bae54515
NFTLaunchpad	63f318d9227938071e73d080571552fd8941fe880e46ec7489d2c46419366d45

As the Ancient8 team has decided not to publish the source code to protect their intellectual property, the users should compare the bytecode hashes compiled by the 0.8.17 compiler version with the smart contracts deployed before interacting with them to make sure that they are the same as the contracts audited.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at https://inspex.gitbook.io/testing-guide/.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

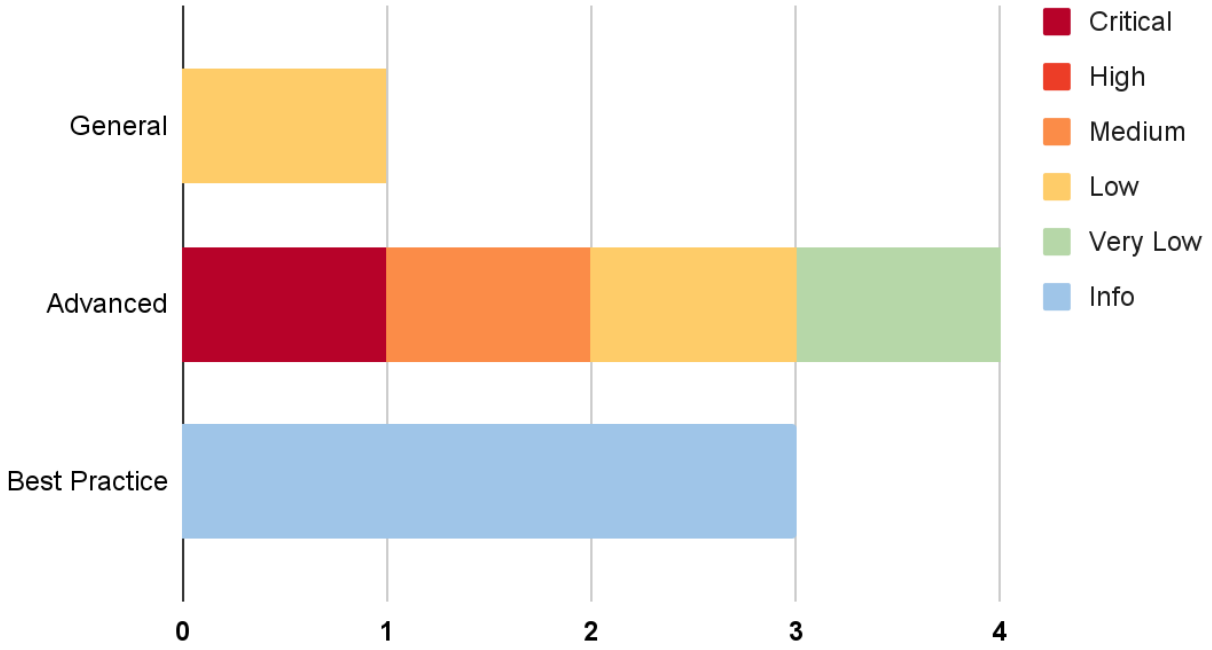
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

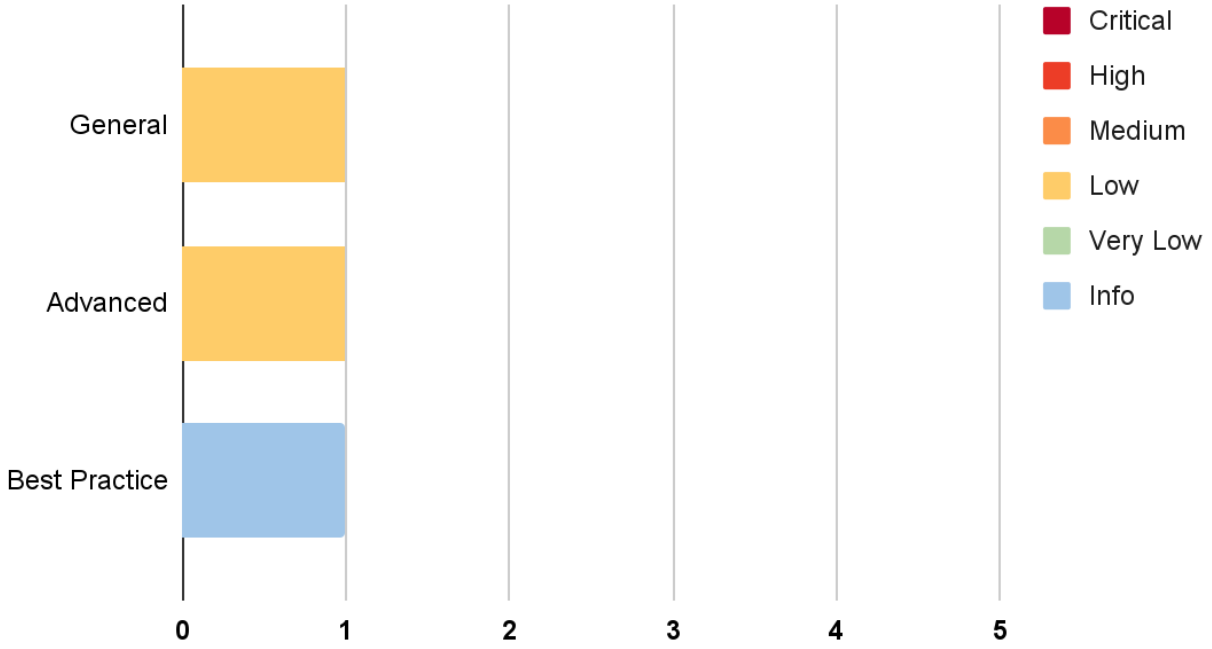
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Missing Access Control	Advanced	Critical	Resolved
IDX-002	Improper msg.value Validation	Advanced	Medium	Resolved
IDX-003	Design Flaw in buyNft() Function	Advanced	Low	Acknowledged
IDX-004	Smart Contract with Unpublished Source Code	General	Low	Acknowledged
IDX-005	Unbound Configuration Parameter in the NFT contract	Advanced	Very Low	Resolved
IDX-006	Inexplicit Solidity Compiler Version	Best Practice	Info	No Security Impact
IDX-007	Insufficient Logging for Privileged Functions	Best Practice	Info	Resolved
IDX-008	Improper Function Visibility	Best Practice	Info	Resolved

* The mitigations or clarifications by Ancient8 can be found in Chapter 5.

5. Detailed Findings Information

5.1. Missing Access Control

ID	IDX-001
Target	NFT NFTLaunchpad
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** Any user can call the `createNFT()` and `addWhiteList()` functions to mint the NFT or add any address to the whitelist. **Likelihood: High** Any user can perform this action without restriction.
Status	**Resolved** The Ancient8 team has resolved this issue by implementing access control for `createNFT()` and `addWhiteList()` functions. Furthermore, the team has added a constraint to prevent adding addresses to the whitelist after the sale time begins.

5.1.1. Description

In the `NFT` contract, the `createNFT()` and `addWhiteList()` functions are used to mint the new NFT and add addresses to the whitelist.

NFT.sol

```
64  function createNFT(
65    string memory tokenURI,
66    address creatorAddress
67  ) public payable returns (uint256) {
68    _tokenIds.increment();
69    uint256 newTokenId = _tokenIds.current();
70
71    _mint(creatorAddress, newTokenId);
72    _setTokenURI(newTokenId, tokenURI);
73
74    _transfer(creatorAddress, address(this), newTokenId);
75
76    totalSupply += 1;
```

```
77
78    return newTokenId;
79  }
```

NFT.sol

```
145 function addWhiteList(address[] memory addresses) public {
146
147   for (uint i = 0; i < addresses.length; i++) {
148     if (!isAddressInwhiteList(addresses[i])) {
149       whiteList.push(addresses[i]);
150     }
151   }
152 }
```

The `whiteList` state will be used to verify users in private and combination sale periods in the `buyNft()` function.

NFT.sol

```
84  function buyNft() public payable {
85    uint256 timestamp = block.timestamp;
86
87    if (saleType == SaleType.whiteListSale) {
88      require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
89
90      bool withinWhiteList = isAddressInwhiteList(msg.sender);
91
92      require(withinWhiteList, 'Only white list user can buy white list NFT');
93    }
94
95    if (saleType == SaleType.publicSale) {
96      require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
    can be bought after public sale time');
97    }
98
99    if (saleType == SaleType.combinationSale) {
100     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
101
102     if (timestamp < publicSaleTime) {
103       bool withinWhiteList = isAddressInwhiteList(msg.sender);
104
105       require(withinWhiteList, 'Only white list user can buy white list NFT');
106     }
107   }
108
```

```
109   require(purchasedAmount < totalSupply, 'Purchased amount should be smaller
      than total supply');
110
111   require(msg.value >= nftPrice, 'Input price should be equal or more than NFT
      price');
112
113   purchasedAmount += 1;
114
115   _transfer(address(this), msg.sender, purchasedAmount);
116
117   payable(publisherAddress).transfer(msg.value);
118 }
```

Without access control, any user can call these functions to arbitrarily mint the NFT or add the whitelist as desired, and this will break the business logic.

5.1.2. Remediation

An access control measure must be implemented to prevent unauthorized actors from using the functions, and the `whiteList` state should be set only once by the authorized account to prevent arbitrarily setting it after the sale period begins.

Inspex suggests modifying the `createNftBox()` function to parse the `nftOwner` and `whiteListAddress` parameters when creating the `NFT` contract, for example:

NftLaunchpad.sol

```
29 function createNftBox(
30   uint256 price,
31   uint saleType,
32   uint256 whiteListSaleTime,
33   uint256 publicSaleTime,
34   uint256 saleEndTime,
35   address publisherAddress,
36   address nftOwner,
37   address[] calldata whiteListAddress
38 ) public returns (address) {
39   require(
40     owner == msg.sender,
41     'Only admin can create NFT Box'
42   );
43
44   NFT newNftBox = new NFT(price, saleType, whiteListSaleTime, publicSaleTime,
     saleEndTime, publisherAddress, nftOwner, whiteListAddress);
45
46   address addr = address(newNftBox);
47   nftBoxes.push(addr);
48
```

```
49    return addr;
50  }
```

After that, calling the new `addWhiteList()` function in the `constructor()` function in the `NFT` contract to set the `whiteList` and new `owner` states, and modifying any relateable state or function as shown below:

NFT.sol

```
33  mapping (address => bool) whiteList;
34  constructor(
35    uint256 _price,
36    uint _saleType,
37    uint256 _whiteListSaleTime,
38    uint256 _publicSaleTime,
39    uint256 _saleEendTime,
40    address _publisherAddress,
41    address _owner,
42    address[] memory _whiteListAddress
43  ) ERC721('NFT', 'NFT') {
44    owner = _owner;
45
46    nftPrice = _price;
47
48    if (_saleType == 0) {
49      saleType = SaleType.publicSale;
50    } else if (_saleType == 1) {
51      saleType = SaleType.whiteListSale;
52    } else if (_saleType == 2) {
53      saleType = SaleType.combinationSale;
54    }
55
56    whiteListSaleTime = _whiteListSaleTime;
57    publicSaleTime = _publicSaleTime;
58    saleEndTime = _saleEendTime;
59    publisherAddress = _publisherAddress;
60    addWhiteList(_whiteListAddress);
61  }
```

NFT.sol

```
145 function addWhiteList(address[] memory addresses) public {
146   for (uint i = 0; i < addresses.length; i++) {
147     whiteList[addresses[i]] = true;
148   }
149 }
150
151 /**
152 * @notice check if an address is in the white list
```

```
153 * @param addr the address to be checked
154 */
155 function isAddressInwhiteList(address addr) public view returns (bool) {
156   return whiteList[addr];
157 }
158
159 /**
160 * @notice check if current account is in the white list
161 */
162 function isInWhiteList() external view returns (bool) {
163   address addr = msg.sender;
164   return whiteList[addr];
165 }
```

Moreover, adding access control to the `createNFT()` function, for example:

NFT.sol

```
64 function createNFT(
65   string memory tokenURI,
66   address creatorAddress
67 ) external payable returns (uint256) {
68   require(msg.sender == owner, "Only owner");
69   _tokenIds.increment();
70   uint256 newTokenId = _tokenIds.current();
71
72   _mint(creatorAddress, newTokenId);
73   _setTokenURI(newTokenId, tokenURI);
74
75   _transfer(creatorAddress, address(this), newTokenId);
76
77   totalSupply += 1;
78
79   return newTokenId;
80 }
```

Please note that the remediation for other issues are not yet applied in the examples above.

5.2. Improper msg.value Validation

ID	IDX-002
Target	NFT
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Medium **Impact:** High If the user sets the `msg.value` greater than the NFT price when calling the `buyNft()` function, the user will lose the excess native token and cannot get it back. Thus, it is unfair to the user and will damage the platform's reputation. **Likelihood:** Low It is unlikely that the users will intentionally or accidentally set `msg.value` to exceed the NFT price.
Status	Resolved The Ancient8 team has resolved this issue as suggested by modifying the validation check.

5.2.1. Description

In the `NFT` contract, the user can call the `buyNft()` function to buy the NFT from the contract by validating that the `msg.value` is equal to or greater than the NFT price.

NFT.sol

```
84  function buyNft() public payable {
85    uint256 timestamp = block.timestamp;
86
87    if (saleType == SaleType.whiteListSale) {
88      require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
89
90      bool withinWhiteList = isAddressInwhiteList(msg.sender);
91
92      require(withinWhiteList, 'Only white list user can buy white list NFT');
93    }
94
95    if (saleType == SaleType.publicSale) {
96      require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
    can be bought after public sale time');
97    }
98
99    if (saleType == SaleType.combinationSale) {
```

```
100     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
101
102     if (timestamp < publicSaleTime) {
103       bool withinWhiteList = isAddressInwhiteList(msg.sender);
104
105       require(withinWhiteList, 'Only white list user can buy white list NFT');
106     }
107   }
108
109   require(purchasedAmount < totalSupply, 'Purchased amount should be smaller
    than total supply');
110
111   require(msg.value >= nftPrice, 'Input price should be equal or more than NFT
    price');
112
113   purchasedAmount += 1;
114
115   _transfer(address(this), msg.sender, purchasedAmount);
116
117   payable(publisherAddress).transfer(msg.value);
118 }
```

However, if the `msg.value` is greater than the price, the excess native token will be transferred to the `publisherAddress` address. Resulting in unfairness to the user and will cause reputation loss to the platform.

5.2.2. Remediation

Inspex suggests modifying the validation check in the `buyNft()` function at line 111, for example:

NFT.sol

```
84  function buyNft() public payable {
85    uint256 timestamp = block.timestamp;
86
87    if (saleType == SaleType.whiteListSale) {
88      require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
89
90      bool withinWhiteList = isAddressInwhiteList(msg.sender);
91
92      require(withinWhiteList, 'Only white list user can buy white list NFT');
93    }
94
95    if (saleType == SaleType.publicSale) {
96      require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
    can be bought after public sale time');
```

```
97    }
98
99    if (saleType == SaleType.combinationSale) {
100     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
101
102     if (timestamp < publicSaleTime) {
103       bool withinWhiteList = isAddressInwhiteList(msg.sender);
104
105       require(withinWhiteList, 'Only white list user can buy white list NFT');
106     }
107   }
108
109   require(purchasedAmount < totalSupply, 'Purchased amount should be smaller
    than total supply');
110
111   require(msg.value == nftPrice, 'Input price should be equal NFT price');
112
113   purchasedAmount += 1;
114
115   _transfer(address(this), msg.sender, purchasedAmount);
116
117   payable(publisherAddress).transfer(msg.value);
118 }
```

Please note that the remediation for other issues are not yet applied in the examples above.

5.3. Design Flaw in buyNft() Function

ID	IDX-003
Target	NFT
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium The users can buy a specific NFT to gain the valuable ones in the `buyNft()` functions. Thus, it is unfair to the other users and may cause a reputation loss to the platform. **Likelihood:** Low It is possible to buy a specific NFT. Moreover, anyone that monitors the transaction pool can front-run the legitimate users to acquire the valuable ones. However, the value of the NFTs does not make this profitable because each NFT has nearly the same rarity. With low profit, there is low motivation.
Status	**Acknowledged** The Ancient8 team has acknowledged the issue by adjusting the buy process to get the NFT index by randomizing instead of the order since the team wants the process to finish in a transaction.

5.3.1. Description

In the `NFT` contract, the `buyNft()` function is used to buy the NFT. The NFT received will be the same order as the order of NFTs that was created by the `NFT` contract, which is provided by the contract as shown below:

NFT.sol

```
84  function buyNft() public payable {
85    uint256 timestamp = block.timestamp;
86
87    if (saleType == SaleType.whiteListSale) {
88      require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
89
90      bool withinWhiteList = isAddressInwhiteList(msg.sender);
91
92      require(withinWhiteList, 'Only white list user can buy white list NFT');
93    }
94
95    if (saleType == SaleType.publicSale) {
96      require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
```

```
can be bought after public sale time');
  }

  if (saleType == SaleType.combinationSale) {
    require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT only can be bought after white list sale time');

    if (timestamp < publicSaleTime) {
      bool withinWhiteList = isAddressInwhiteList(msg.sender);

      require(withinWhiteList, 'Only white list user can buy white list NFT');
    }
  }

  require(purchasedAmount < totalSupply, 'Purchased amount should be smaller than total supply');

  require(msg.value >= nftPrice, 'Input price should be equal or more than NFT price');

  purchasedAmount += 1;

  _transfer(address(this), msg.sender, purchasedAmount);

  payable(publisherAddress).transfer(msg.value);
}
```

With the current design, the NFT that is provided by the contract is already revealed (`tokenURI`) in the `createNFT()` function as shown below:

NFT.sol

```
function createNFT(
  string memory tokenURI,
  address creatorAddress
) public payable returns (uint256) {
  _tokenIds.increment();
  uint256 newTokenId = _tokenIds.current();

  _mint(creatorAddress, newTokenId);
  _setTokenURI(newTokenId, tokenURI);

  _transfer(creatorAddress, address(this), newTokenId);

  totalSupply += 1;

  return newTokenId;
}
```

This provides an opportunity for the users to buy a specific token id, resulting in unfairness to the other users.

5.3.2. Remediation

Inspex suggests redesigning the buying process to get the NFT token id by using secure randomness, such as Chainlink VRF (https://docs.chain.link/docs/vrf/v2/introduction), instead of an order of the token id, for example:

NFT.sol

```
1  import { VRFCoordinatorV2Interface } from
   "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
2  import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";
3
4  contract NFT is VRFConsumerBaseV2, ERC721URIStorage, ReentrancyGuard {
```

NFT.sol

```
84  VRFCoordinatorV2Interface private vrfCoordinator;
85
86  // Your subscription ID.
87  uint64 s_subscriptionId; // Initital value in constructor
88  bytes32 keyHash; // Initital value in constructor
89  uint32 callbackGasLimit = 500000; // Initital value in constructor
90
91  // The default is 3, but you can set this higher.
92  uint16 requestConfirmations = 3; // Initital value in constructor
93
94  uint256[] tokenId;
95  mapping(uint256 => address) buyAddress;
96
97  function createNFT(
98    string memory tokenURI,
99    address creatorAddress
100 ) public payable returns (uint256) {
101   _tokenIds.increment();
102   uint256 newTokenId = _tokenIds.current();
103
104   _mint(creatorAddress, newTokenId);
105   _setTokenURI(newTokenId, tokenURI);
106
107   _transfer(creatorAddress, address(this), newTokenId);
108
109   totalSupply += 1;
110
111   tokenId.push(newTokenId);
112   return newTokenId;
```

```
113 }
114
115 /**
116 * @notice buy NFT from the NFT Box
117 */
118 function buyNft() public payable {
119   uint256 timestamp = block.timestamp;
120
121   if (saleType == SaleType.whiteListSale) {
122     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
123
124     bool withinWhiteList = isAddressInwhiteList(msg.sender);
125
126     require(withinWhiteList, 'Only white list user can buy white list NFT');
127   }
128
129   if (saleType == SaleType.publicSale) {
130     require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
    can be bought after public sale time');
131   }
132
133   if (saleType == SaleType.combinationSale) {
134     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
    only can be bought after white list sale time');
135
136     if (timestamp < publicSaleTime) {
137       bool withinWhiteList = isAddressInwhiteList(msg.sender);
138
139       require(withinWhiteList, 'Only white list user can buy white list NFT');
140     }
141   }
142
143   require(purchasedAmount < totalSupply, 'Purchased amount should be smaller
    than total supply');
144
145   require(msg.value >= nftPrice, 'Input price should be equal or more than NFT
    price');
146
147   purchasedAmount += 1;
148
149   payable(publisherAddress).transfer(msg.value);
150
151   uint256 requestId = vrfCoordinator.requestRandomWords(keyHash,
    vrfSubscriptionId, requestConfirmations, callbackGasLimit, 1);
152
153   buyAddress[requestId] = msg.sender;
```

```
154 }
155
156 /// @notice Chainlink's callback function to fulfill the randomness
157 /// @param _randomWords The random results from VRF
158 function fulfillRandomWords(
159   uint256 _requestId,
160   uint256[] memory _randomWords
161 ) internal override {
162   require(buyAddress[_requestId] != address(0), "Request not found");
163
164   uint256 randomIndex = _randomWords[0] % tokenId.length;
165   uint256 randomTokenId = tokenId[randomIndex];
166   address to = buyAddress[_requestId];
167
168   tokenId[randomIndex] = tokenId[tokenId.length - 1];
169   tokenId.pop();
170   delete buyAddress[_requestId];
171
172   _transfer(address(this), to, randomTokenId);
173 }
```

Alternatively, using the blockhash of future block as a random source, for example:

NFT.sol

```
84 struct RequestData {
85    uint256 randomBlock;
86    uint256 futureBlock;
87    address user;
88 }
89 mapping(uint256 => RequestData) request;
90 uint256 requestId = 0;
91
92 function buyNft() public payable {
93   uint256 timestamp = block.timestamp;
94
95   if (saleType == SaleType.whiteListSale) {
96     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
   only can be bought after white list sale time');
97
98     bool withinWhiteList = isAddressInwhiteList(msg.sender);
99
100    require(withinWhiteList, 'Only white list user can buy white list NFT');
101  }
102
103  if (saleType == SaleType.publicSale) {
104    require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only
   can be bought after public sale time');
```

```
105   }
106
107   if (saleType == SaleType.combinationSale) {
108     require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT
only can be bought after white list sale time');
109
110     if (timestamp < publicSaleTime) {
111       bool withinWhiteList = isAddressInwhiteList(msg.sender);
112
113       require(withinWhiteList, 'Only white list user can buy white list NFT');
114     }
115   }
116
117   require(purchasedAmount < totalSupply, 'Purchased amount should be smaller
than total supply');
118
119   require(msg.value >= nftPrice, 'Input price should be equal or more than NFT
price');
120
121   purchasedAmount += 1;
122
123   request[requestId] = RequestData(block.number, block.number + 1, msg.sender);
124   requestId += 1;
125
126   payable(publisherAddress).transfer(msg.value);
127 }
128
129 function claimNft(uint256 _requestId) external {
130   RequestData storage data = request[_requestId];
131
132   require(data.user != address(0), "Request not exists.");
133
134   if(block.number - data.futureBlock < 256){
135     uint256 randomIndex = uint256(blockhash(data.futureBlock)) %
tokenIds.length;
136     uint256 randomTokenId = tokenIds[randomIndex];
137
138     tokenIds[randomIndex] = tokenIds[tokenIds.length - 1];
139     tokenIds.pop();
140
141     _transfer(address(this), data.user, randomTokenId);
142   } else {
143     purchasedAmount -= 1 ;
144   }
145
146   delete request[_requestId];
147 }
```

Please note that the remediation for other issues are not yet applied in the examples above.

5.4. Smart Contract with Unpublished Source Code

ID	IDX-004
Target	NFT NFTLaunchpad
Category	General Smart Contract Vulnerability
CWE	CWE-1006: Bad Coding Practices
Risk	**Severity:** Low **Impact:** Medium The logic of the smart contract may not align with the user's understanding, causing undesired actions to be taken when the user interacts with the smart contract. **Likelihood:** Low The possibility for the users to misunderstand the functionalities of the contract is not very high with the help of the documentation and user interface.
Status	**Acknowledged** The Ancient8 team has acknowledged this issue and decided not to publish the source code because the team wants to protect their intellectual property.

5.4.1. Description

The smart contract source code is not publicly published, so the users will not be able to easily verify the correctness of the functionalities and the logic of the smart contract by themselves. Therefore, it is possible that the user's understanding of the smart contract does not align with the actual implementation, leading to undesired actions on interacting with the smart contract.

5.4.2. Remediation

Inspex suggests publishing the contract source code through a public code repository or verifying the smart contract source code on the blockchain explorer so that the users can easily read and understand the logic of the smart contract by themselves.

5.5. Unbound Configuration Parameter in the NFT contract

ID	IDX-005
Target	NFT
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Very Low **Impact:** Low When the contract owner sets the `saleEndTime` state less than current time, the users can not use the `buyNft()` function. However, the contract owner can create a new `NFT` contract to set the `saleEndTime` state. **Likelihood:** Low These affected states can be set intentionally and accidentally, but only the owner of the `NFTLaunchpad` contract has the right to set them.
Status	Resolved The Ancient8 team has resolved this issue by adding the boundaries of `_whiteListSaleTime`, `_publicSaleTime` and `_saleEendTime` parameters to the `NFT` contract's `constructor()` function.

5.5.1. Description

In the `NFTLaunchpad` contract, the `createNftBox()` function allows the owner to create a new `NFT` contract and store the `NFT` address in the `nftBoxes` state.

The contract owner can set the sale type of NFT and the period of sell time by setting the `whiteListSaleTime`, `publicSaleTime` and `saleEndTime` parameters at line 41.

NftLaunchpad.sol

```
28 function createNftBox(
29   uint256 price,
30   uint saleType,
31   uint256 whiteListSaleTime,
32   uint256 publicSaleTime,
33   uint256 saleEndTime,
34   address publisherAddress
35 ) public returns (address) {
36   require(
37     owner == msg.sender,
38     'Only admin can create NFT Box'
39   );
```

```
40
41    NFT newNftBox = new NFT(price, saleType, whiteListSaleTime, publicSaleTime,
      saleEndTime, publisherAddress);
42
43    address addr = address(newNftBox);
44    nftBoxes.push(addr);
45
46    return addr;
47  }
```

In the NFT contract, the _whiteListSaleTime, _publicSaleTime and _saleEendTime parameters will be set into whiteListSaleTime, publicSaleTime and saleEndTime states in lines 53-55, respectively.

Without any boundaries, the saleEndTime state can be set to less than the current time, and the whiteListSaleTime or publicSaleTime states can be set over than saleEndTime state.

NFT.sol

```
33  constructor(
34    uint256 _price,
35    uint _saleType,
36    uint256 _whiteListSaleTime,
37    uint256 _publicSaleTime,
38    uint256 _saleEendTime,
39    address _publisherAddress
40  ) ERC721('NFT', 'NFT') {
41    owner = payable(msg.sender);
42
43    nftPrice = _price;
44
45    if (_saleType == 0) {
46      saleType = SaleType.publicSale;
47    } else if (_saleType == 1) {
48      saleType = SaleType.whiteListSale;
49    } else if (_saleType == 2) {
50      saleType = SaleType.combinationSale;
51    }
52
53    whiteListSaleTime = _whiteListSaleTime;
54    publicSaleTime = _publicSaleTime;
55    saleEndTime = _saleEendTime;
56    publisherAddress = _publisherAddress;
57  }
```

If the saleEndTime state is less than the current time or the whiteListSaleTime and publicSaleTime states are greater than the saleEndTime state, the buyNft() function will not be able to execute due to validation check at lines 88, 96 and 100.

NFT.sol

```
//solidity:M4RK_TH1S:100:4+124,88:4+124,96:4+117
function buyNft() public payable {
  uint256 timestamp = block.timestamp;

  if (saleType == SaleType.whiteListSale) {
    require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT only can be bought after white list sale time');

    bool withinWhiteList = isAddressInwhiteList(msg.sender);

    require(withinWhiteList, 'Only white list user can buy white list NFT');
  }

  if (saleType == SaleType.publicSale) {
    require(timestamp >= publicSaleTime && timestamp < saleEndTime, 'NFT only can be bought after public sale time');
  }

  if (saleType == SaleType.combinationSale) {
    require(timestamp >= whiteListSaleTime && timestamp < saleEndTime, 'NFT only can be bought after white list sale time');

    if (timestamp < publicSaleTime) {
      bool withinWhiteList = isAddressInwhiteList(msg.sender);

      require(withinWhiteList, 'Only white list user can buy white list NFT');
    }
  }

  require(purchasedAmount < totalSupply, 'Purchased amount should be smaller than total supply');

  require(msg.value >= nftPrice, 'Input price should be equal or more than NFT price');

  purchasedAmount += 1;

  _transfer(address(this), msg.sender, purchasedAmount);

  payable(publisherAddress).transfer(msg.value);
}
```

5.5.2. Remediation

Inspex suggests adding the boundary to the `constructor()` function in `NFT` contract to guarantee the possible value of the `_whiteListSaleTime`, `_publicSaleTime` and `_saleEendTime` parameters will always be within the acceptable range in lines 45-56, for example:

NFT.sol

```
33  constructor(
34    uint256 _price,
35    uint _saleType,
36    uint256 _whiteListSaleTime,
37    uint256 _publicSaleTime,
38    uint256 _saleEendTime,
39    address _publisherAddress
40  ) ERC721('NFT', 'NFT') {
41    owner = payable(msg.sender);
42
43    nftPrice = _price;
44
45    if (_saleType == 0) {
46      saleType = SaleType.publicSale;
47      require(_publicSaleTime < _saleEendTime, "Public sale time must be less
    than sale end time.");
48    } else if (_saleType == 1) {
49      saleType = SaleType.whiteListSale;
50      require(_whiteListSaleTime < _saleEendTime, "White list sale time must be
    less than sale end time.");
51    } else if (_saleType == 2) {
52      saleType = SaleType.combinationSale;
53      require(_whiteListSaleTime < _publicSaleTime, "White list sale time must be
    less than public sale time.");
54      require(_publicSaleTime < _saleEendTime, "Public sale time must be less
    than sale end time.");
55    }
56    require(block.timestamp < _saleEendTime);
57
58    whiteListSaleTime = _whiteListSaleTime;
59    publicSaleTime = _publicSaleTime;
60    saleEndTime = _saleEendTime;
61    publisherAddress = _publisherAddress;
62  }
```

Please note that the remediation for other issues are not yet applied in the examples above.

5.6. Inexplicit Solidity Compiler Version

ID	IDX-006
Target	NFT NFTLaunchpad
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact The Ancient8 team has acknowledged this issue and leaving this right now has no direct impact.

5.6.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

NFT.sol

```
1 // SPDX-License-Identifier: Apache-2.0
2 pragma solidity ^0.8.9;
```

The affected contracts are listed in the table below:

File	Version
NFT.sol (L: 2)	^0.8.9
NftLaunchpad.sol (L: 2)	^0.8.9

5.6.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is 0.8.17 (https://github.com/ethereum/solidity/releases), for example:

NFT.sol

```
1 // SPDX-License-Identifier: Apache-2.0
2 pragma solidity 0.8.17;
```

5.7. Insufficient Logging for Privileged Functions

ID	IDX-007
Target	NFTLaunchpad
Category	Smart Contract Best Practice
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Ancient8 team has resolved this issue by emitting events for the execution of privileged functions.

5.7.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions.

For example, the contract owner can create the new `NFT` contract by executing the `createNftBox()` function in the `NFTLaunchpad` contract, and no events are emitted.

NftLaunchpad.sol

```
28  function createNftBox(
29    uint256 price,
30    uint saleType,
31    uint256 whiteListSaleTime,
32    uint256 publicSaleTime,
33    uint256 saleEndTime,
34    address publisherAddress
35  ) public returns (address) {
36    require(
37      owner == msg.sender,
38      'Only admin can create NFT Box'
39    );
40
41    NFT newNftBox = new NFT(price, saleType, whiteListSaleTime, publicSaleTime,
    saleEndTime, publisherAddress);
42
43    address addr = address(newNftBox);
44    nftBoxes.push(addr);
45
```

```
46    return addr;
47  }
```

5.7.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

NftLaunchpad.sol

```
28  event CreateNftBox(address _newNFTBox);
29  function createNftBox(
30    uint256 price,
31    uint saleType,
32    uint256 whiteListSaleTime,
33    uint256 publicSaleTime,
34    uint256 saleEndTime,
35    address publisherAddress
36  ) public returns (address) {
37    require(
38      owner == msg.sender,
39      'Only admin can create NFT Box'
40    );
41
42    NFT newNftBox = new NFT(price, saleType, whiteListSaleTime, publicSaleTime,
    saleEndTime, publisherAddress);
43
44    address addr = address(newNftBox);
45    nftBoxes.push(addr);
46    emit CreateNftBox(addr);
47
48    return addr;
49  }
```

Please note that the remediation for other issues are not yet applied in the examples above.

5.8. Improper Function Visibility

ID	IDX-008
Target	NFT NFTLaunchpad
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Ancient8 team has resolved this issue as suggested by changing all functions' visibility to `external` if they are not called from any internal function.

5.8.1. Description

Public functions that are never called internally by the contract itself should have external visibility. This improves the readability of the contract, allowing clear distinction between functions that are externally used and functions that are also called internally.

For example, the following source code shows that the `getNftBoxAddresses()` function of the NFTLaunchpad contract is set to public and it is never called from any internal function.

NftLaunchpad.sol

```
53 function getNftBoxAddresses() public view returns (address[] memory) {
54   return nftBoxes;
55 }
```

The following table contains all functions that have public visibility and are never called from any internal function.

File	Contract	Function
NFT.sol (L: 64)	NFT	createNFT()
NFT.sol (L: 84)	NFT	buyNft()
NFT.sol (L: 123)	NFT	getTotalSupply()
NFT.sol (L: 130)	NFT	getNftPrice()

NFT.sol (L: 137)	NFT	getPurchasedAmount()
NFT.sol (L: 145)	NFT	addWhiteList()
NFT.sol (L: 171)	NFT	isInWhiteList()
NftLaunchpad.sol (L: 29)	NFTLaunchpad	createNftBox()
NftLaunchpad.sol (L: 53)	NFTLaunchpad	getNftBoxAddresses()

5.8.2. Remediation

Inspex suggests changing all functions' visibility to `external` if they are not called from any internal function as shown in the following example:

NftLaunchpad.sol

```
53 function getNftBoxAddresses() external view returns (address[] memory) {
54   return nftBoxes;
55 }
```

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

inspeX

CYBERSECURITY PROFESSIONAL SERVICE