Space3

Smart Contract Audit Report
Prepared for Ancient8

Date Issued:	Mar 24, 2023
Project ID:	AUDIT2023007
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2023007
Version	v1.0
Client	Ancient8
Project	Space3
Auditor(s)	Wachirawit Kanpanluk Kongkit Chatchawanhirun
Author(s)	Wachirawit Kanpanluk
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Mar 24, 2023	Full report	Wachirawit Kanpanluk

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. Use of Upgradable Contract Design 10
5.2. Arbitrary initialize() Function Call in the Implementation Contract 11
5.3. Inexplicit Solidity Compiler Version 13
5.4. Improper Function Visibility 14
6. Appendix **16**
6.1. About Inspex 16

1. Executive Summary

As requested by Ancient8, Inspex team conducted an audit to verify the security posture of the Space3 smart contracts on Mar 14, 2023. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Space3 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 high, and 3 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that Space3 smart contracts have high-level protections in place to be safe from most attacks.

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 - Space3 is the contract that allows the Ancient8 to easily create new NFTs for sale on the marketplace as well as create NFT badges to incentivize users on the bounty system.

Scope Information:

Project Name	Space3
Website	https://ancient8.gg
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	Token, NFT

Audit Information:

Audit Method	Whitebox
Audit Date	Mar 14, 2023
Reassessment Date	Mar 23, 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 following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 01d9db31c91dd4c725549cd0837f2dcf75259b05)

Contract	Location (URL)
Space3	https://github.com/ancient8-gg/Space3-NFT1155/blob/01d9db31c9/contracts/A8-NFT-1155.sol
A8NFT	https://github.com/ancient8-gg/Space3-NFT1155/blob/01d9db31c9/contracts/A8-NFT.sol

Reassessment: (Commit: 7d93185df9adc2b581b78af9454186b975468acd)

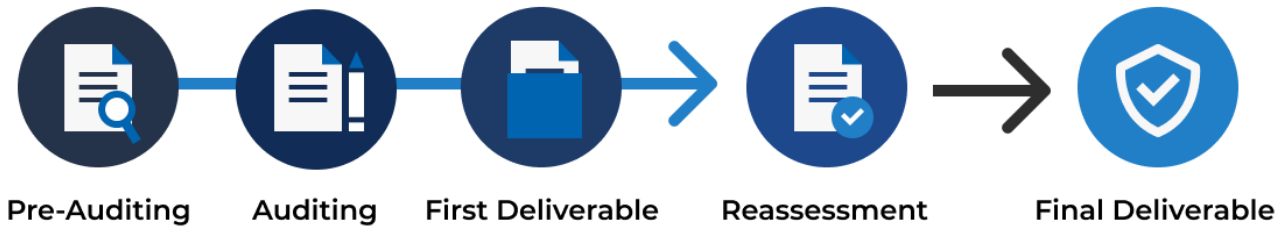
Contract	Location (URL)
Space3	https://github.com/ancient8-gg/Space3-NFT1155/blob/7d93185df9/contracts/A8-NFT-1155.sol
A8NFT	https://github.com/ancient8-gg/Space3-NFT1155/blob/7d93185df9/contracts/A8-NFT.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

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**.

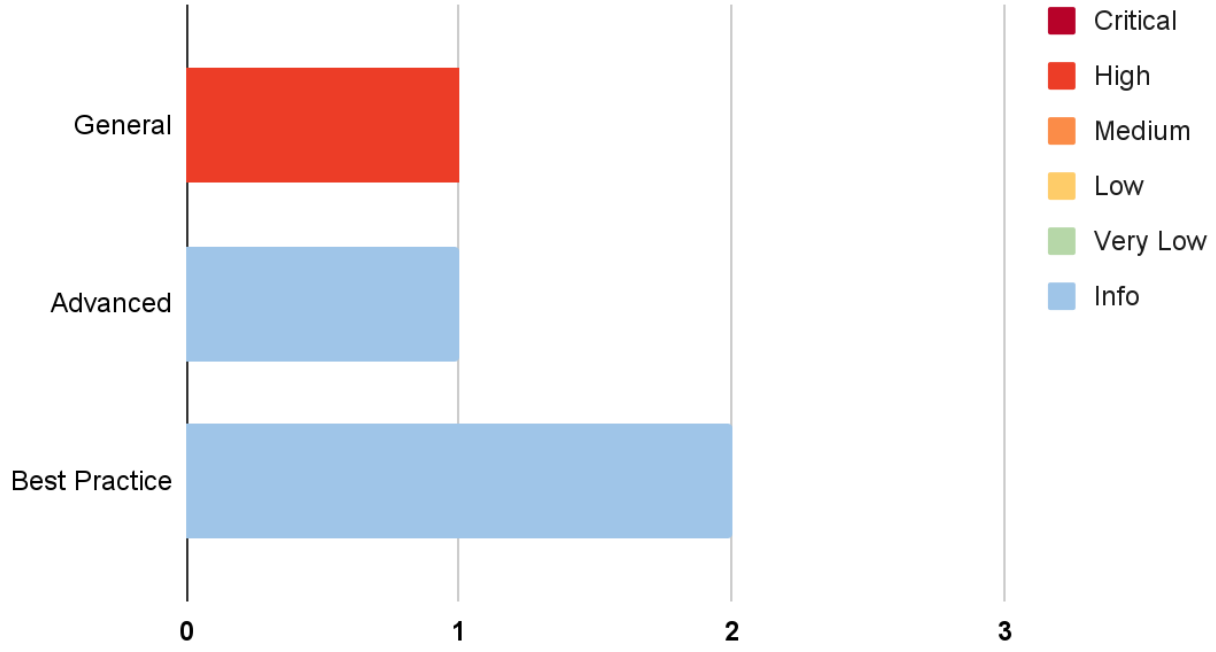
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

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

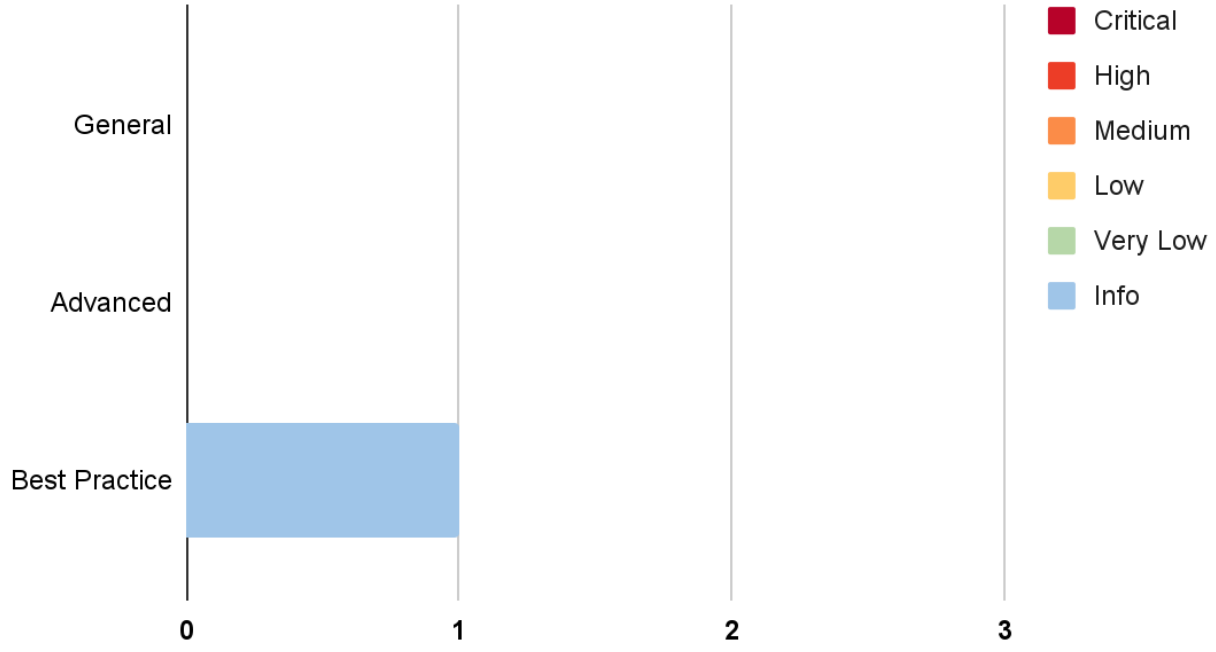
4. Summary of Findings

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	Use of Upgradable Contract Design	General	High	Resolved
IDX-002	Arbitrary initialize() Function Call in the Implementation Contract	Advanced	Info	Resolved
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	Info	No Security Impact
IDX-004	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. Use of Upgradable Contract Design

ID	IDX-001
Target	Space3
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the users' funds anytime. **Likelihood:** Medium This action can be performed by the proxy owner without any restriction.
Status	Resolved The Ancient8 team has resolved this issue by removing the proxy pattern that can make the smart contracts upgradeable in commit `7d93185df9adc2b581b78af9454186b975468acd`.

5.1.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As these smart contracts are upgradable, the logic of them can be modified by the owner anytime, making the smart contracts untrustworthy.

5.1.2. Remediation

Inspex suggests deploying the contracts without the proxy pattern or any solution that can make the smart contracts upgradeable.

However, if upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes at least 24 hours on the proxy owner role. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.2. Arbitrary initialize() Function Call in the Implementation Contract

ID	IDX-002
Target	Space3
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Ancient8 team has resolved this issue due to the removal of the proxy pattern in commit `7d93185df9adc2b581b78af9454186b975468acd`.

5.2.1. Description

In the `Space3` contract, several roles are granted to the caller when the contract is set up with the `initialize()` function.

A8-NFT-1155.sol

```
32 function initialize() public initializer {
33     __ERC1155_init("");
34     __AccessControl_init();
35     __ERC1155Supply_init();
36     __UUPSUpgradeable_init();
37
38     _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
39     _grantRole(MINTER_ROLE, msg.sender);
40     _grantRole(UPGRADER_ROLE, msg.sender);
41 }
```

Normally the `Space3` contract will be deployed as an implementation contract, the proxy contract will be deployed, and the `initialize()` function should be called immediately.

Even so, the `initialize()` function in `Space3` contract can be arbitrarily called by anyone, resulting in the caller grants roles of the implementation contract.

However, calling the `initialize()` function in the implementation contract will not affect the storage in the proxy contract. Therefore, there is no direct security impact.

5.2.2. Remediation

Inspex suggests calling the `_disableInitializers()` function from the OpenZepplin Initializable contract in the `constructor()` function.

By calling the `_disableInitializers()` function in the `constructor()` function, this prevents initialization of the implementation contract itself, for example:

A8-NFT-1155.sol

```
28 constructor() {
29     _disableInitializers();
30 }
```

5.3. Inexplicit Solidity Compiler Version

ID	IDX-003
Target	Space3 A8NFT
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. However, there is no security impact on the platform.

5.3.1. 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:

A8-NFT.sol

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

The affected contracts are listed in the table below:

File	Version
A8-NFT-1155.sol (L: 2)	^0.8.9
A8-NFT.sol (L: 2)	^0.8.9

5.3.2. 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 0.8.19 (https://github.com/ethereum/solidity/releases), for example:

A8-NFT.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.19;
```

5.4. Improper Function Visibility

ID	IDX-004
Target	Space3 A8NFT
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 the issue by changing the visibility of functions that are not called from any internal function to external in commit `7d93185df9adc2b581b78af9454186b975468acd`.

5.4.1. 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.

The following source code shows that the `safeMint()` function of the `A8NFT` contract is set to public and it is never called from any internal function.

A8-NFT.sol

```
17 function safeMint(address to, string memory uri) public onlyOwner {
18     uint256 tokenId = _tokenIdCounter.current();
19     _tokenIdCounter.increment();
20     _safeMint(to, tokenId);
21     _setTokenURI(tokenId, uri);
22 }
```

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

File	Contract	Function
A8-NFT-1155.sol (L:32)	Space3	initialize()
A8-NFT-1155.sol (L:43)	Space3	mint()

A8-NFT-1155.sol (L:57)	Space3	getCurrentTokenId()
A8-NFT.sol (L:17)	A8NFT	safeMint()

5.4.2. Remediation

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

A8-NFT.sol

```
17 function safeMint(address to, string memory uri) external onlyOwner {
18     uint256 tokenId = _tokenIdCounter.current();
19     _tokenIdCounter.increment();
20     _safeMint(to, tokenId);
21     _setTokenURI(tokenId, uri);
22 }
```

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