QuillAudits

Audit Report
May, 2023

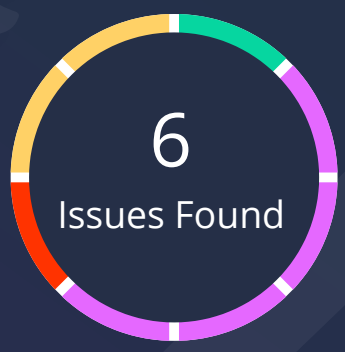
For

ASTRO CHIMPZ™

Table of Content

Executive Summary

Project Name	AstroChimz Scenes
Overview	The contracts are designed to mint non-fungible tokens (NFTs) to its users. The project integrates the standard Openzeppelin library, using the ERC721RoyaltyUpgradable and OwnableUpgradable to help handle the creation of NFTs, easily track token royalties information, and aid in the ownership management.
Timeline	17 May, 2023 - 22 May, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Language	Solidity
Blockchain	Ethereum
Scope of Audit	The scope of this audit was to analyze AstroChimz Scenes codebase for quality, security, and correctness.
	https://github.com/VishalWaman/studioKulture/tree/main/contracts **Branch:** Main **Commit:** d766c8cca34e114fc270a4f24d83804f06c2ef1d
Contracts in Scope	**SceneContractV1** and **SceneStorage**
Fixed In	0f220c365a0ffdd362de129b48a5afddef11479c



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Resolved Issues	1	1	2	2

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - SceneContractV1.sol

High Severity Issues

1. UpdateData Function can be Called by Any User to Divert Sales Fee from Original CollectionWalletAddress

```
function updateData(
    uint256 _max_count,
    uint256 _reveal_time,
    uint256 _price,
    address _collection_address
) external virtual {
    maxMintCount = _max_count;
    revealTime = _reveal_time;
    tokenPrice = _price;
    CollectionWalletAddress = _collection_address;
}
```

Description

the updateData function allows for the changing of some critical state variables in the contract. Variables that could be changed with the function includes the maxMintCount, revealTime, tokenPrice, and collectionWalletAddress; all these variables aids in moderating the maximum amount to mint per user, the revealing time after mint, the price of minting a token and the address that receives the purchasing fee. This function was intended to allow only the owner of the contract to call this function and set these values but any user can call this function to increase the number of mints that can be called per mint and also change the collectionWalletAddress to theirs to divert the funds to their addresses. Attacker could as well set this address to address zero and funds raised from sales goes to the null address and is irredeemable.

Remediation

Add an onlyOwner modifier to the updateData function to prevent any user from calling the function. Also add an input validation to check that the input values; for instance the collectorWalletAddress cannot be set to address zero to prevent sending sales fee to an irredeemable address.

Status

Resolved

Medium Severity Issues

2. UpdateDefaultURI Function has no Modifier to Regulate who Update the URIs Info

```
function updateDefaultURI(
    string memory _base,
    string memory _extension,
    string memory _unreveal_uri
) external virtual {
    baseUri = _base;
    baseExtension = _extension;
    unRevealURI = _unreveal_uri;
}
```

Description

The updateDefaultURI function sets critical variables in relation to the project URIs. However, this function can be called by any address because the function does not have a modifier that regulates who must call this function.

Remediation

It is recommended to add a modifier to the function to allow only the project address to update all these critical state variables.

Status

Resolved

Low Severity Issues

3. Missing Zero Address Check

Description

The collectionWalletAddress state variable holds the address that receives payments made when users call the sale function. Although, it was mentioned earlier in the first issue that this variable alongside others, could be set by anybody due to the absence of a modifier to the updateData function. However, adding just a modifier still leaves the function susceptible to changing the collectionWalletAddress to address zero. It is important to check that when an address is passed as a parameter to the function, it should revert to prevent being set to the null address.

Remediation

Add a null address prevention check to the updateData function.

Status

Resolved

4. Missing Event Emission for Critical State Changes

Description

Event emission is important to project because it aids in onchain tracking on when critical state changes happen at the contract. When the owner of the contract updates some critical state variables, it is important to emit these changes. It is also recommended that these events are indexed. Functions to emit an event are as follows:

- updateDefaultURIs
- updateData
- updateSaleStatus

Remediation

Emit events in these functions and also ensure that other events in the contract are indexed. *Reference.*

Status

Resolved

Informational Issues

5. Inconsistency with the Code and Comment

Description

The code comments describing the input parameters of the updateSaleStatus is inconsistent with the parameters in the function. It shares the same comment with the updateData function.

```
/**
 * @dev owner can on and off the public and presale
 *
 * @param _status max count of total scenes to be minted.
 * @param _max_per_wallet reveal original metadata time.
 *
 * Requirements:
 * - msg.sender must be owner of contract.
 *
 */
ftrace | funcSig
function updateSaleStatus(
    uint256 _status,
    uint256 _max_per_wallet
) external onlyOwner {
    saleStatus = _status;
    maxPerWallet = _max_per_wallet;
}
```

Remediation

Update the comment to precisely describe the _status and _max_per_wallet parameters.

Status

Resolved

6. No Check to Prevent Whitelisting or Blacklisting an Address Twice

Description

The whitelistAddress and blacklistAddress function gives the contract owner the privilege of adding an array of addresses to either whitelist or blacklist. However, one likely issue could be when an address appears twice in the array. When a situation like that happens, this will allow the owner to set them twice or more, depending on how much it appears within the array, causing the owner to expend more gas on the whitelist.

Remediation

Add a check in the function to check that when a person has been previously whitelisted or blacklisted, it should not be done twice in order to save gas.

Status

Resolved

Client's comment: Added the double assign status check and zero address check on both functionalities.

Functional Testing

Some of the tests performed are mentioned below:

- Should get the token details after the contract is deployed
- Should confirm that the contract can be initialized just once.
- Should confirm that the contract can be upgraded with newer features
- Should allow onlyOwner calls some critical functions to update state variables
- Should allow the owner to set the default and specific royalty information.
- Should confirm that the owner can whitelist addresses.
- Should confirm that the owner can blacklist addresses.
- Should confirm that only whitelisted addresses can call sale during presale
- Should confirm that whitelisted addresses can purchase more than the max mint amount.
- Should prevent blacklisted addresses from transferring out tokens from their addresses.
- Should verify that a single address when repeated twice in the array to be whitelisted or blacklisted, can be done twice or more.
- Should check that the owner successfully airdrop tokens into addresses.
- Should verify that the token uri is revealed after the revealedTime.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#117)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#126)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result
 of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
SceneContractV1.sale(uint256).i (contracts/SceneContractV1.sol#186) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#434-456) ignores re
turn value by IERC721ReceiverUpgradeable(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.s
ol#441-452)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
SceneContractV1.updateSaleStatus(uint256,uint256) (contracts/SceneContractV1.sol#89-95) should emit an event for:
        - saleStatus = _status (contracts/SceneContractV1.sol#93)
        - maxPerWallet = _max_per_wallet (contracts/SceneContractV1.sol#94)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
SceneContractV1.updateData(uint256,uint256,uint256,address)._collection_address (contracts/SceneContractV1.sol#71) lacks a zero-check on :
                - CollectionWalletAddress = _collection_address (contracts/SceneContractV1.sol#76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
SceneContractV1.tokenURI(uint256) (contracts/SceneContractV1.sol#249-267) uses timestamp for comparisons
        Dangerous comparisons:
        - revealTimestamp[_tokenId] + revealTime > block.timestamp (contracts/SceneContractV1.sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#434-456) uses assem
bly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448-450)
AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#18-38) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#24-26)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#30-32)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#66-70)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#86-93)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#100-109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.18', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.18 (contracts/SceneContractV1.sol#3)
        - 0.8.18 (contracts/SceneStorage.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721RoyaltyUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/common/ERC2981Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
SceneContractV1.airdrop(address[]) (contracts/SceneContractV1.sol#146-155) has costly operations inside a loop:
        - tokenCounter += 1 (contracts/SceneContractV1.sol#150)
SceneContractV1.sale(uint256) (contracts/SceneContractV1.sol#168-202) has costly operations inside a loop:
        - tokenCounter += 1 (contracts/SceneContractV1.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4) allows old versions
```


```
SceneContractV1.tokenURI(uint256) (contracts/SceneContractV1.sol#249-267) uses timestamp for comparisons
        Dangerous comparisons:
        - revealTimestamp[_tokenId] + revealTime > block.timestamp (contracts/SceneContractV1.sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#434-456) uses assem
bly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448-450)
AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#18-38) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#24-26)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#30-32)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#66-70)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#86-93)
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#100-109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.18', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.18 (contracts/SceneContractV1.sol#3)
        - 0.8.18 (contracts/SceneStorage.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721RoyaltyUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/common/ERC2981Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
SceneContractV1.airdrop(address[]) (contracts/SceneContractV1.sol#146-155) has costly operations inside a loop:
        - tokenCounter += 1 (contracts/SceneContractV1.sol#150)
SceneContractV1.sale(uint256) (contracts/SceneContractV1.sol#168-202) has costly operations inside a loop:
        - tokenCounter += 1 (contracts/SceneContractV1.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4) allows old versions
```

Summary

In this report, we have considered the security of AstroChimz Scenes. We performed our audit according to the procedure described above.

Some issues of high, medium, low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In The End, AstroChimz Scenes Team resolved all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the AstroChimz Scenes Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the AstroChimz Scenes Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$16B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2023

For

ASTRO CHIMPZ™

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com