Cyberscope

Audit Report

Web3Punks

December 2023

Network	BSC
Address	0x29b43DA747BA6Ce57887727d6FB3FA1a2fb2ff53
Audited by	© cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x29b43da747ba6ce5788 7727d6fb3fa1a2fb2ff53

Audit Updates

Initial Audit	12 Dec 2023

Source Files

Filename	SHA256
contracts/W3PContract.sol	1526f2832ccad0918c46d0bff9e00670541 87ce712673e52b0002cf26474f1b2
@openzeppelin/contracts/utils/Strings.sol	cb2df477077a5963ab50a52768cb74ec6f3 2177177a78611ddbbe2c07e2d36de
@openzeppelin/contracts/utils/Context.sol	b2cfee351bcafd0f8f27c72d76c054df9b57 1b62cfac4781ed12c86354e2a56c
@openzeppelin/contracts/utils/Address.sol	8b85a2463eda119c2f42c34fa3d942b61ae e65df381f48ed436fe8edb3a7d602
@openzeppelin/contracts/utils/math/SignedMath.sol	420a5a5d8d94611a04b39d6cf5f0249255 2ed4257ea82aba3c765b1ad52f77f6
@openzeppelin/contracts/utils/math/SafeMath.sol	fc16aa4564878e1bb65740239d0c142245 1cd32136306626ac37f5d5e0606a7b
@openzeppelin/contracts/utils/math/Math.sol	85a2caf3bd06579fb55236398c1321e15fd 524a8fe140dff748c0f73d7a52345
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b30 64325801d73654847a5fb11c58b1e5

@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	77f0f7340c2da6bb9edbc90ab6e7d3eb8e2ae18194791b827a3e8c0b11a09b43
@openzeppelin/contracts/token/ERC721/IERC721.sol	c8d867eda0fd764890040a3644f5ccf5db92f852779879f321ab3ad8b799bf97
@openzeppelin/contracts/token/ERC721/ERC721.sol	7af3ff063370acb5e1f1a2aab125ceca457cd1fa60ff8afa37aabc366349d286
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	f16b861aa1f623ccc5e173f1a82d8cf45b678a7fb81e05478fd17eb2ccb7b37e
@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol	7bf559fad1068a1329517b56b1ecddefa67e79a03bb0801b9e6bf06bf73eb334
@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol	e04aa070ad6f111fae49b96a056671f36307a93dd79b27612e72560e4a9749b2
@openzeppelin/contracts/security/Pausable.sol	2072248d2f79e661c149fd6a6593a8a3f038466557c9b75e50e0b001bcb5cf97
@openzeppelin/contracts/interfaces/IERC721.sol	e3bcee0ce85a310031fcef279f963e73c12c676a66c5c562ab3945ccf10aecff
@openzeppelin/contracts/interfaces/IERC4906.sol	6b572852b6d6e1db371287a0eb443a724e9005e025025b9c82ebc8804433c0ff
@openzeppelin/contracts/interfaces/IERC165.sol	410e40cd79f1b82bb6bbab95fa4279252cae6e3962b0bff46ab4855f6de91d35

Overview

This document provides the overview of the smart contract audit conducted for the "Web3Punks" contract. This contract is designed for minting NFTs with various attributes and dynamic pricing mechanisms. It utilizes ERC721 standards and leverages OpenZeppelin libraries for enhanced security and functionality. The contract owner has the authority to pause/unpause the mint of NFTs, change price models, and change critical parameters, which pose several centralization risks that warrant attention.

Functionality

Mint

Users can mint NFTs by providing a token ID, URI, and attributes. Minting is subject to the contract not being paused and adheres to max supply limits.

Dynamic Pricing

The contract incorporates a dynamic pricing mechanism based on the token ID and attributes. It includes different base prices for various ranges of token IDs and attribute counts.

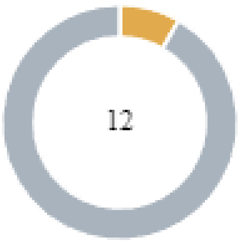
Mint Limit Enforcement

Implements a mint limit logic based on the token ID threshold, ensuring controlled minting activity.

Mint Limit Individually

Implements a mint limit logic for each user individually, where they can either mint 1 or 7 NFTs, based on how many total NFTs have been already minted.

Findings Breakdown



Critical		0
Medium		1
Minor / Informative		11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical   Medium   Minor / Informative

Severity	Code	Description	Status
Medium	OLIE	Ownership Limits Inadequate Enforcement	Unresolved
Minor / Informative	CCR	Contract Centralization Risks	Unresolved
Minor / Informative	PRE	Potential Reentrance Exploit	Unresolved
Minor / Informative	VAR	Variable Assignment Redundancy	Unresolved
Minor / Informative	COI	Conditional Operators Inefficiency	Unresolved
Minor / Informative	IVS	Inefficient Variable Scope	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	L06	Missing Events Access Control	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

OLIE - Ownership Limits Inadequate Enforcement

Criticality	Medium
Location	contracts/W3PContract.sol#L116
Status	Unresolved

Description

The contract contains a limit on the maximum number of NFTs that can be minted. According to the implementation of the limit, the user is prevented from minting new NFTs if the balance is equal to the threshold. A user can bypass the minting limit by receiving additional NFTs through transfers, effectively accumulating more than the intended limit.

```
// Mint limit enforcement
if (balanceOf(to) == limit) {
    revert MintLimitReached({message: "Mint Limit Reached",
limitValue: limit});
}
```

Recommendation

The team is advised to check if the user's balance is greater or equal to the threshold. This way users will not be able to mint NFTs even if they have more NFTs than the threshold.

CCR - Contract Centralization Risks

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L79,159,178,191,204,211
Status	Unresolved

Description

The contract owner has the authority to pause/unpause the mint of NFTs, change price models, and change critical parameters like max supply. While this configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on this type of configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function pause() public onlyOwner {
        _pause();
}
function updateBasePrice1e3(uint256 basePrice1k) external
onlyOwner {
        basePrice1e3 = basePrice1k;
}
function updateMaxSupply(uint256 newMaxSupply) external
onlyOwner {
        maxSupply = newMaxSupply;
}
```

Recommendation

To mitigate these centralization risks, consider the following strategies:

- Implement a governance mechanism that allows NFT holders to vote on critical decisions.
- Transition control from a single owner to a multi-signature wallet.
- Implement time locks for critical functions.

PRE - Potential Reentrance Exploit

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L127
Status	Unresolved

Description

The `safeMint` function is susceptible to a potential reentrance exploit due to the sequence of operations where external calls are made before updating the contract's state. The function call `_safeMint`, which could interact with an external contract, followed by state changes. During the reentrance phase, an NFT with the same `URI` could be produced. As a result, the entire business logic of the contract might be violated since the uniqueness of the `URI` will be broken.

```
// Minting process
totalNFTsMinted++;
_safeMint(to, tokenId);
_setTokenURI(tokenId, uri);
_mintedURIs[uri] = true;
tokenAttributes[tokenId] = attributes;
emit NFTMinted(to, tokenId, attributes);
payable(mintFeeReceiver).transfer(price);
```

Recommendation

The team is advised to prevent the potential re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Add lockers/mutexes in the method scope. It is important to note that mutexes do not prevent cross-function reentrancy attacks.
- Proceed with the external call as the last statement of the method, so that the state will have been updated properly during the re-entrance phase.

VAR - Variable Assignment Redundancy

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L113
Status	Unresolved

Description

In the `safeMint` function, there is an observed redundancy in the assignment of the `limit` variable. This variable is assigned twice in immediate succession with the same conditional logic, which is unnecessary and can be erased for better code clarity and efficiency.

```
// Pricing logic
(tokenId < MINT_THRESHOLD)
    ? ((tokenId < MINT_THRESHOLD / 2) ? price = basePrice1e3 :
price = BASE_PRICE_2E3)
    : price = calculatePrice(attributes);
(tokenId < MINT_THRESHOLD) ? limit = 1 : limit = 7;

// Mint limit logic
(tokenId < MINT_THRESHOLD) ? limit = 1 : limit = 7;
```

Recommendation

To enhance the clarity and maintainability of the `safeMint` function, consider eliminating the second instance of the `limit` variable assignment. Since the assignment is identical to the first, it serves no functional purpose and can be safely removed.

COI - Conditional Operators Inefficiency

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L107
Status	Unresolved

Description

The `safeMint` function uses nested conditional (ternary) operators to set the `price` and `limit` variables. This implementation, although functionally correct, poses challenges in terms of code readability, maintainability and performance, since The use of nested ternary operators makes the code difficult to read and understand at a glance and adds unnecessary complexity. Additionally, nested conditional operators are not efficient regarding performance.

```
(tokenId < MINT_THRESHOLD)
            ? ((tokenId < MINT_THRESHOLD / 2) ? price =
basePrice1e3 : price = BASE_PRICE_2E3)
            : price = calculatePrice(attributes);
        (tokenId < MINT_THRESHOLD) ? limit = 1 : limit = 7;
```

Recommendation

To improve the readability, maintainability, efficiency and overall clarity of the `safeMint` function, consider refactoring the `pricing` and `limit` logic by replacing the nested conditional operators with `if-else` statements.

IVS - Inefficient Variable Scope

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L52,53
Status	Unresolved

Description

The `limit` and `price` variables are declared at the contract level as private variables but are used exclusively within the scope of the `safeMint` function. This implementation leads to the following inefficiencies:

- As contract-level variables, `limit` and `price` are stored in contract storage, which is more expensive in terms of gas costs compared to memory. Storage variables are written to the blockchain, incurring higher gas fees, especially when their values are frequently modified.
- Every modification to a storage variable is a state change on the blockchain, which is more expensive and permanent. Since `limit` and `price` are only relevant within a single function call, their state does not need to persist beyond the execution of `safeMint` .

```
uint256 private limit;
uint256 private price;
```

Recommendation

In order to optimize the contract for gas efficiency and improve code clarity, consider declaring `limit` and `price` as local variables within the `safeMint` function. This change ensures that these variables are stored in memory, not in contract storage, thus reducing gas costs associated with storage operations.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L159,178,190,204,211,225
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
basePrice1e3 = basePrice1k;
basePrice = basePriceAttribute;
zeroAttributeBasePrice = basePriceZeroAttribute;
maxSupply = newMaxSupply;
mintFeeReceiver = mintAmountReceiver;
owner = newOwner;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/W3PContract.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L213,226
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
mintFeeReceiver = mintAmountReceiver
owner = newOwner
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L160,179,192,205
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
basePrice1e3 = basePrice1k
basePrice = basePriceAttribute
zeroAttributeBasePrice = basePriceZeroAttribute
maxSupply = newMaxSupply
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L61,226
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
mintFeeReceiver = _mintFeeReceiver
owner = newOwner
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/W3PContract.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.9;
```

Recommendation

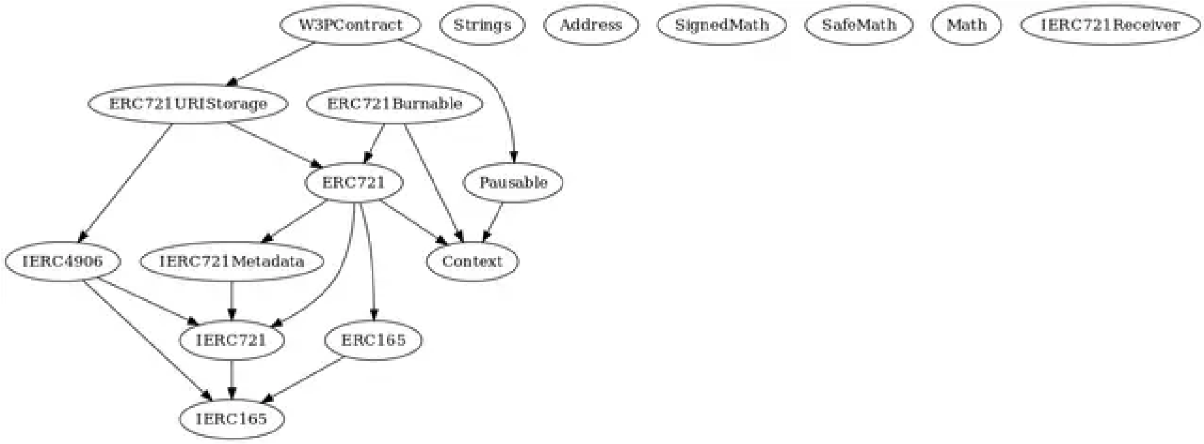
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

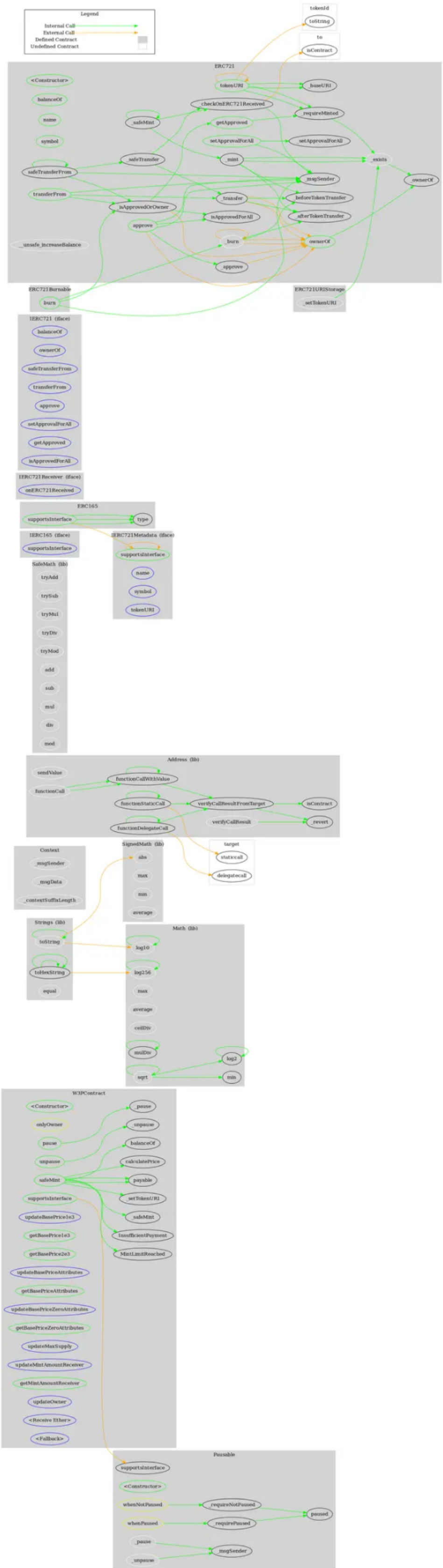
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
W3PContract	Implementation	ERC721URIStorage, Pausable		
		Public	✓	ERC721
	supportsInterface	Public		-
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	safeMint	Public	Payable	whenNotPaused
	calculatePrice	Internal		
	updateBasePrice1e3	External	✓	onlyOwner
	getBasePrice1e3	Public		-
	getBasePrice2e3	Public		-
	updateBasePriceAttributes	External	✓	onlyOwner
	getBasePriceAttributes	Public		-
	updateBasePriceZeroAttributes	External	✓	onlyOwner
	getBasePriceZeroAttributes	Public		-
	updateMaxSupply	External	✓	onlyOwner
	updateMintAmountReceiver	External	✓	onlyOwner
	getMintAmountReceiver	Public		-
	updateOwner	External	✓	onlyOwner

		External	Payable	-
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

Web3Punks contract implements a nft mechanism. It allows users to mint NFTs with diverse attributes and dynamic pricing strategies. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io