Cyberscope

Audit Report

ZODIAC PUPPY

November 2023

Network	MATIC
Address	0x1f93d8952b80a819ba16957f6682c7d94b8d3add
Audited by	© cyberscope

Table of Contents

Review

Contract Name	ZodiacPuppy
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://polygonscan.com/address/0x1f93d8952b80a819ba16957f6682c7d94b8d3add
Address	0x1f93d8952b80a819ba16957f6682c7d94b8d3add
Network	MATIC
Symbol	ZP
Decimals	6

Audit Updates

Initial Audit	17 Nov 2023

Source Files

Filename	SHA256
ZodiacPuppy.sol	36b2988d32b61be7dc87998f0be376be13513144f87b57fb97af2ca221c42c41
UpdatableOperatorFilterer.sol	a3a5306db46f386babba148d5c8176f18123285753e2a6f5b3f6c9fa69539c60
Strings.sol	8f6058629388fb90c8fff4c078c693c5fa145692e5d3bf3a65d8cad375fb7874

RevokableOperatorFilterer.sol	e852ad54ffd40ea17b06a6e4fc1208e78292c2df1688179677a5be33214ab2b2
RevokableDefaultOperatorFilterer.sol	e5ac4500da2848fa6b48092de84a129fbbee25b4eaa4dd8ccc87997d08a769a9
OwnedRegistrant.sol	58badb651c38397c6f45828e1c859c5efdb2fa21060de697cf62a56d38dc5310
Ownable2Step.sol	d9c7d62d66d28886abb7023e302438d7c7b06509a169b42e28b398cedcf66244
Ownable.sol	cb4c67c6d164ed15c6cfc466507a7f3bac219b5f0d73093e2d5718827b71a4d4
OperatorFilterer.sol	7c73600c7f7a76453436cf61a60ea9073500bcbcfec9d8fc45ef748fdf5845c5
OperatorFilterRegistryErrorsAndEvents.sol	910b9bddd571b66b50b48f305389b2f0711dea240b6b828d9d610c96025e4dec
OperatorFilterRegistry.sol	40e17ce52c79887c118140815fb6d40b269771b5a9b7a3f032cdc9c62ffebed3
IOperatorFilterRegistry.sol	6bdda9746ecd096eebf8b9186682b800ed9ee5f3ea8163fe011cc01fcd54629f
IERC721Receiver.sol	d6df589726d997eb52ba219df5c26f42d2ab78077bbb2def0c34ba7d50ba20a4
IERC721Metadata.sol	a64d7ab8eb146225e91eb00a8a9bdd0d80dc31be9622ee7f3ba00e415fe204d2
IERC721.sol	a540552b4cb854eae45f85dba6b1949bb9734300baef8c3fa1de5e3d50b2a159
IERC2981.sol	03348aa2fb08832518a274e220eb7ea0c5f15a0c4aee5b4aad3ade7e27f60936
IERC20.sol	9be043cb9394f0fdfd7bfa0ae9201bedb08ce930a3da8c2c5cfdae56ae7eb13f

IERC165.sol	48ab1f2a907c12063745ba591bced76c1d306b6436d43ce9ce9b5ddd6c515989
EnumerableSet.sol	041f2c4c3c278410cde6e20f93f0ca61606be31f07616228eccf89a88174d80c
ERC721.sol	06a7dabe0ec2609477dc9724802c642d02adf1e5e594b4a185536363428297fe
ERC2981.sol	bfe74b0ba5f73b96435c25892519c4e2c1b3a90308fb289d725d679c7db98792
ERC165.sol	8ece085beaa27efb8f9774c5e8923d4e76379c8214c5960b5b3ef42ad0ca7230
DefaultOperatorFilterer.sol	22a900cacb9a073970aba6b35d27b8697077f8e2b571db64c9cb47046a5c295e
Counters.sol	e10346b263158ef9aa46f5cbf0432b2f5221fa84f999074a93cc2fa3bd4fbb92
Context.sol	d45b53197201430d256d1fea664c15923ec3ac0c72547dbcc8a2778379ab4d92
Address.sol	9d32f7fe96db01b7edd847fc3633bf993a1c4e19151adc3305360cb50c665543

Overview

The ZodiacPuppy NFT contract represents an ERC721 implementation designed for the creation and management of a unique NFT collection. This contract incorporates functionalities like token-based payments, royalty management through ERC2981, and various owner-controlled features. The contract is designed to handle a maximum supply of 12,000 NFTs, with special provisions for public minting, with a a unique pricing structure divided into different sections. Additionally, the contract contains the `preMintNFTs` function which gives the contract owner the authority to mint NFTs.

Owner Functionalities

The owner of the "ZodiacPuppy" contract has extensive control over its functionalities. These include the ability to:

- Mint NFTs to specific addresses.
- Manage the public mint status, enabling or disabling it as required.
- Set and update various parameters like the base URI, contract URI, public sale cost, maximum NFTs per wallet and transaction, and decimal points for pricing.
- Adjust royalty information, including the royalty address and the percentage of sales allocated as royalties.
- Configure the pricing for different sections of the NFT collection, allowing for dynamic pricing strategies.
- Update the token contract used for payments and set specific wallet for pre-mints and royalties.

Mint Functionality

Users can interact with the "ZodiacPuppy" contract primarily through the mint function. This function allows them to mint a specified number of NFTs within a selected price section, provided they hold enough of the specified ERC20 token used for payment. The minting process involves a check to ensure that the total supply does not exceed the maximum limit and that users adhere to per wallet and per transaction limits. The payment for minting is transferred in tokens from the user to the owner's wallet, aligning with the `publicSaleCost` and the selected price section. Additionally, the contract supports a multi-section pricing structure, offering flexibility in pricing different parts of the NFT collection.

Roles

Owner

The owner can interact with the following functions in the "ZodiacHero" NFT contract:

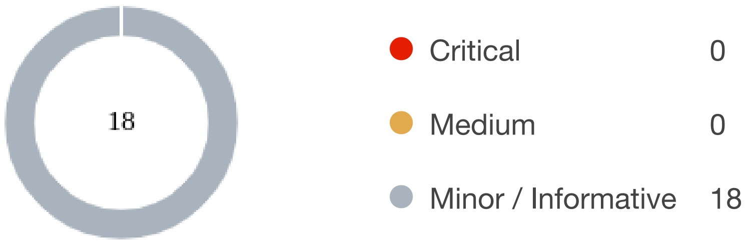
- preMintNFTs(uint256 startingID, uint256 endingID)
- setRoyaltyInfo(address _receiver, uint96 _royaltyFeesInBips)
- setBaseExtension(string memory _newBaseExtension)
- setPublic_mint_status(bool _public_mint_status)
- setPublicSaleCost(uint256 _publicSaleCost)
- setMax_per_wallet(uint256 _max_per_wallet)
- setMax_per_txn(uint256 _max_per_txn)
- setDecimals(uint256 _decimals)
- setRoyaltyAddress(address _royaltyAddress)
- setPremintwallet(address _premintwallet)
- setContractURI(string calldata _contractURI)
- setTokenContract(address _tokenContract)
- setSectionPrice(uint256 sectionNumber, uint256 _price)
- setBulkSetPrice(uint256[] memory _price)

User

Users can interact with the following functions in the "ZodiacHero" NFT contract:

- mint(uint256 _mintAmount, uint256 _priceSection)
- transferFrom(address from, address to, uint256 tokenId)
- tokenURI(uint256 tokenId)

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	18	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
⚪	CCR	Contract Centralization Risk	Unresolved
⚪	MDA	Misleading Decimals Adjustment	Unresolved
⚪	MSB	Max Supply Bypass	Unresolved
⚪	ISA	Inconsistent Supply Allocation	Unresolved
⚪	MC	Missing Check	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	DPI	Decimals Precision Inconsistency	Unresolved
⚪	RPF	Redundant Payable Function	Unresolved
⚪	EIS	Excessively Integer Size	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved
⚪	L14	Uninitialized Variables in Local Scope	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved
●	L22	Potential Locked Ether	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L58,306,313,317,321,325,341
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the owner has exclusive rights to mint tokens to specific address, pause the minting process, set URIs the NFTs, adjust the mint price and the maximum amount per wallet and per transaction. This concentration of power in a single authority presents a centralization risk, potentially undermining the decentralized nature expected in blockchain applications.

Additionally, the `onlyAllowedOperator` and `onlyAllowedOperatorApproval` modifiers, through their reliance on the `OPERATOR_FILTER_REGISTRY` , introduce centralization risks related to control, security, and the efficient operation of the smart contract.

```
   function preMintNFTs(uint256 startingID, uint256 endingID) public
onlyOwner {
        for (uint256 i = startingID; i <= endingID; i++) {
            _safeMint(premintwallet, i);
            totalSupply++;
        }
    }
    function setBaseExtension(string memory _newBaseExtension)
        public
        onlyOwner
    {
        baseExtension = _newBaseExtension;
    }

    function setPublic_mint_status(bool _public_mint_status) public
onlyOwner {
        public_mint_status = _public_mint_status;
    }

    function setPublicSaleCost(uint256 _publicSaleCost) public onlyOwner
{
        publicSaleCost = _publicSaleCost;
    }

    function setMax_per_wallet(uint256 _max_per_wallet) public onlyOwner
{
        max_per_wallet = _max_per_wallet;
    }

    function setMax_per_txn(uint256 _max_per_txn) public onlyOwner {
        max_per_txn = _max_per_txn;
    }

    function setContractURI(string calldata _contractURI) public
onlyOwner {
        contractURI = _contractURI;
    }
   modifier onlyAllowedOperator(address from) virtual {
        // Allow spending tokens from addresses with balance
        // Note that this still allows listings and marketplaces with
escrow to transfer tokens if transferred
        // from an EOA.
        if (from != msg.sender) {
            _checkFilterOperator(msg.sender);
        }
        _;
    }

    modifier onlyAllowedOperatorApproval(address operator) virtual {
        _checkFilterOperator(operator);
        _;
    }

    function _checkFilterOperator(address operator) internal view
virtual {
        // Check registry code length to facilitate testing in
environments without a deployed registry.
        if (address(OPERATOR_FILTER_REGISTRY).code.length > 0) {
            if
(!OPERATOR_FILTER_REGISTRY.isOperatorAllowed(address(this), operator)) {
                revert OperatorNotAllowed(operator);
            }
        }
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MDA - Misleading Decimals Adjustment

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L193
Status	Unresolved

Description

The contract is currently use the `setDecimals` function, which allows for the modification of the `decimals` variable, a crucial parameter that represents the token's decimal precision. This function, accessible only by the contract owner, permits the alteration of the decimal count. However, this approach poses a significant risk as it allows for the setting of a decimal value that may not accurately represent the actual decimal precision of the tokens used by the contract. The potential discrepancy between the set decimal value and the token's inherent decimal structure can lead to confusion and errors in transactions, and calculation impacting the reliability and accuracy of the contract's operations.

```
    function setDecimals(uint256 _decimals) public onlyOwner {
        decimals = _decimals;
    }
```

Recommendation

It is recommended to remove the `setDecimals` function from the contract to prevent the possibility of setting an incorrect `decimal` value. Instead, the contract should directly retrieve the token's decimal count using the `.decimals` function from the token's contract address. This method ensures that the decimal value used within the contract always accurately reflects the actual decimal precision of the token, maintaining consistency and reliability in the contract's operations. By directly referencing the token's inherent decimal structure, the contract eliminates the risk of decimal mismatches and enhances its overall integrity and functionality.

MSB - Max Supply Bypass

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L58
Status	Unresolved

Description

The contract allows the owner to potentially exceed the predefined `MAX_SUPPLY` limit of tokens through the `preMintNFTs` function. This function, which is intended to mint tokens in a range from `startingID` to `endingID`, lacks a validation mechanism to ensure that the total supply of tokens does not surpass the `MAX_SUPPLY` during execution. The absence of such a check in the loop that performs the minting operation can lead to scenarios where the total supply of tokens exceeds the intended limit. This oversight could result in the dilution of token value and breach the trust of the token holders who rely on the predefined supply cap for value assurance.

```
    function preMintNFTs(uint256 startingID, uint256 endingID) public
onlyOwner {
        for (uint256 i = startingID; i <= endingID; i++) {
            _safeMint(premintwallet, i);
            totalSupply++;
        }
    }
```

Recommendation

It is recommended to introduce additional safeguards within the `preMintNFTs` function to prevent minting that exceeds the `MAX_SUPPLY` limit. This can be effectively achieved by adding a check before each minting operation within the loop to ensure that the total minted quantity, when added to the existing totalSupply, will not surpass the `MAX_SUPPLY`. By implementing this safeguard, the integrity of the token's supply limit can be maintained, ensuring adherence to the predefined cap and preserving the trust of stakeholders in the token's value.

ISA - Inconsistent Supply Allocation

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L32,62
Status	Unresolved

Description

The contract is designed with a `MAX_SUPPLY` of `12,000` NFTs. The NFTs are distributed across 24 different `_priceSections` , each with distinct start and end token IDs. However, the non-continuous nature of the start and end token IDs across these sections leads to a situation where, even when all possible `_priceSections` are utilized, the actual NFT supply minted will be lower than the desired `MAX_SUPPLY` value.

This difference is due to gaps between the end token ID of one section and the start token ID of the next. For instance, the first section ends at token ID `200` and the next starts at `502` , as a result there is a gap where no tokens are minted. This pattern is repeated across the sections, resulting in a significant number of token IDs within the `MAX_SUPPLY` range that are never allocated. Consequently, this design flaw restricts the full utilization of the `MAX_SUPPLY` , potentially impacting the project's distribution.

```
   uint256 public MAX_SUPPLY = 12000;
    ...

    function mint(uint256 _mintAmount, uint256 _priceSection)
public payable {
    require(totalSupply + _mintAmount <= MAX_SUPPLY, "Maximum
supply exceeds");

    uint256 startTokenId;
    uint256 endTokenId;

    // Define the start and end token IDs based on the selected
price section
    if (_priceSection == 1) {
        startTokenId = 76;
        endTokenId = 200;
        sectionCount[1] = sectionCount[1] + _mintAmount;
    } else if (_priceSection == 2) {
        startTokenId = 502;
        endTokenId = 1000;
        sectionCount[2] = sectionCount[2] + _mintAmount;
    }
    ...
    } else if (_priceSection == 24) {
        startTokenId = 11502;
        endTokenId = 12000;
        sectionCount[24] = sectionCount[24] + _mintAmount;
    }
    ...

    }
```

Recommendation

It is recommended to reevaluate and update the contract's code to rectify the identified supply allocation inconsistency. Specifically, the contract should effectively use the maximum supply of NFTs, ensuring no portion of the intended supply remains unallocated. To achieve this, the start and end token IDs for each `_priceSection` could be adjusted to create a continuous range without gaps. This crucial realignment will facilitate the full utilization of the `MAX_SUPPLY` of `12,000` NFTs, ensuring that all token IDs within the targeted range are allocated. By implementing these changes, the contract will better align with its intended distribution strategy, eliminating the current shortfall and maximizing the efficiency of the token distribution process.

MC - Missing Check

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L65,317,350
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically:

- `_priceSection` in mint and `sectionNumber` in `setSectionPrice` : The contract uses the `_priceSection` and `sectionNumber` variables, which are intended to be set up to 24 as indicated by the code of the `mint` function. However, in the `mint` and `setSectionPrice` functions, there are no checks in place to prevent setting `_priceSection` and `sectionNumber` , respectively, to values higher than 24. This oversight could result in operations that exceed the intended bounds, leading to potential logical errors or inconsistencies in the contract.
- `setMax_per_wallet` and `setMax_per_txn` Limits: The `setMax_per_wallet` and `setMax_per_txn` functions allow the contract owner to set the respective limits to values that could exceed the `MAX_SUPPLY` . This absence of constraint checks could lead to inconsistencies and unintended behavior in the token distribution process, such as allowing a single wallet or transaction to exceed the total token supply.
- Length of `_price` in `setBulkSetPrice` : The `setBulkSetPrice` function lacks a check to ensure the length of the `_price` array does not exceed 24, which is the total number of sections. Without this validation, there is a risk of passing an array with a length greater than 24, potentially leading to array out-of-bounds issues or other logical inconsistencies.

```
   function mint(uint256 _mintAmount, uint256 _priceSection)
public payable {
    require(totalSupply + _mintAmount <= MAX_SUPPLY, "Maximum
supply exceeds");
    ...
    }

    function setPublicSaleCost(uint256 _publicSaleCost) public
onlyOwner {
        publicSaleCost = _publicSaleCost;
    }

    function setMax_per_wallet(uint256 _max_per_wallet) public
onlyOwner {
        max_per_wallet = _max_per_wallet;
    }

    function setMax_per_txn(uint256 _max_per_txn) public
onlyOwner {
        max_per_txn = _max_per_txn;
    }

    function setSectionPrice(uint256 sectionNumber, uint256
_price) public onlyOwner {
        sectionPrice[sectionNumber] = _price;
    }

    function setBulkSetPrice(uint256[] memory _price) public
onlyOwner {
        for(uint256 x = 1; x <= 24; x++){
        sectionPrice[x] = _price[x-1];

        }
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. To address these vulnerabilities and ensure the contract behaves as intended, the following modifications are recommended:

- `_priceSection` and `sectionNumber` Validations: Introduce checks in the `mint` and `setSectionPrice` functions to ensure that the `_priceSection` parameter and `sectionNumber`, respectively, are less than or equal to 24. This can be done by adding require statements that validate these conditions and revert the transaction if they are not met.

- Limit Validations for `setMax_per_wallet` and `setMax_per_txn`: Modify the `setMax_per_wallet` and `setMax_per_txn` functions to include validation checks that prevent setting the `max_per_wallet` and `max_per_txn` limits above the `MAX_SUPPLY`. This can be achieved with require statements that compare the input values against `MAX_SUPPLY` and revert if they exceed it.
- Length Check for `_price` in `setBulkSetPrice`: Implement a check at the beginning of the `setBulkSetPrice` function to ensure that the length of the `_price` array does not exceed 24. This validation will prevent the function from processing arrays that are larger than the total number of sections, maintaining the contract's logical consistency and preventing potential errors.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L317
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
   function setPublicSaleCost(uint256 _publicSaleCost) public
onlyOwner {
        publicSaleCost = _publicSaleCost;
    }

    function setMax_per_wallet(uint256 _max_per_wallet) public
onlyOwner {
        max_per_wallet = _max_per_wallet;
    }

    function setMax_per_txn(uint256 _max_per_txn) public
onlyOwner {
        max_per_txn = _max_per_txn;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L178
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
    require(token.balanceOf(msg.sender) >= (publicSaleCost * 10
** decimals) * _mintAmount, "You do not have enough tokens to
perform the transaction");
     token.transferFrom(msg.sender, owner(), (publicSaleCost *
10 ** decimals) * _mintAmount);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

RPF - Redundant Payable Function

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L62
Status	Unresolved

Description

The contract's function `mint` have the `payable` modifier which means that the users pay with the native token. The function does not use the `msg.value` variable anywhere inside the function to indicate the usage of the payable modifier. As a result, the payable modifier is redundant.

```
    function mint(uint256 _mintAmount, uint256 _priceSection) public
payable {
        require(totalSupply + _mintAmount <= MAX_SUPPLY, "Maximum supply
exceeds");
        ...
    }
```

Recommendation

The team is advised to remove the `payable` modifier from the function's declaration as it is not needed.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L32,65,329,350,355
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Additionally, the variables `_priceSection` , `_decimals` , `sectionNumber` , and the loop variable `uint256 x` inside the functions, are declared using `uint256` . Since the expected range of these variables is limited, utilizing `uint256` is overly excessive and contributes further to the inefficiencies mentioned above.

```
uint256 public MAX_SUPPLY = 12000;
 ...
 uint256 public max_per_wallet = 12000;
 uint256 public max_per_txn = 12000;
 ...
 uint256 public decimals = 6;
 ...
 function mint(..., uint256 _priceSection)
 ...
 function setDecimals(uint256 _decimals)
 ...
 function setSectionPrice(uint256 sectionNumber,...)
 ..
 uint256 x = 1;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L32
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public MAX_SUPPLY = 12000
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L30,32,34,35,41,65,258,275,285,306,313,317,321,325,329,333,337,341,345,350,354
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
bool public public_mint_status = true
uint256 public MAX_SUPPLY = 12000
uint256 public max_per_wallet = 12000
uint256 public max_per_txn = 12000
address public token_Contract =
0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174
uint256 _priceSection
uint256 _mintAmount
address _receiver
uint96 _royaltyFeesInBips
uint256 _salePrice
string memory _newBaseExtension

function setPublic_mint_status(bool _public_mint_status) public
onlyOwner {
        public_mint_status = _public_mint_status;
    }

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L263
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.

```
royaltyFeesInBips = _royaltyFeesInBips
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L290
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
return (_salePrice / 10000) * royaltyFeesInBips
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L68,69
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 startTokenId
uint256 endTokenId
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L262,334,338,347
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.

```
royaltyAddress = _receiver
royaltyAddress = _royaltyAddress
premintwallet = _premintwallet
token_Contract = _tokenContract
```

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	ZodiacPuppy.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 ^0.8.13;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L179
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transferFrom(msg.sender, owner(), (publicSaleCost * 10 **
decimals) * _mintAmount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	ZodiacPuppy.sol#L65
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
function mint(uint256 _mintAmount, uint256 _priceSection)
public payable {
    require(totalSupply + _mintAmount <= MAX_SUPPLY, "Maximum
supply exceeds");

    uint256 startTokenId;
    uint256 endTokenId;

...
        } else {
            revert("Token ID not within the selected price
section");
        }

    }

}
```

Recommendation

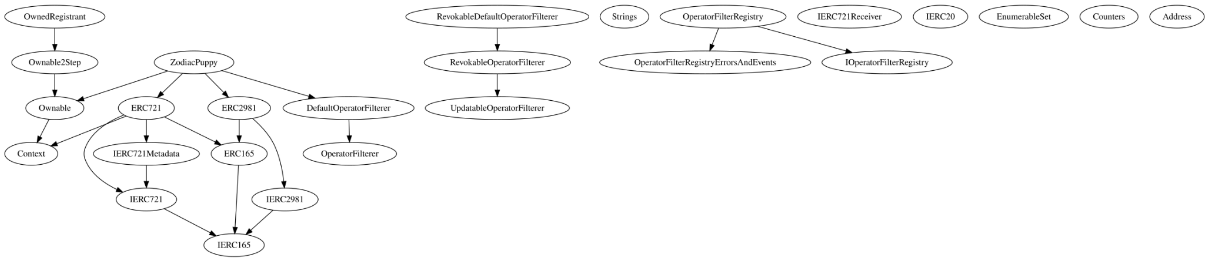
The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

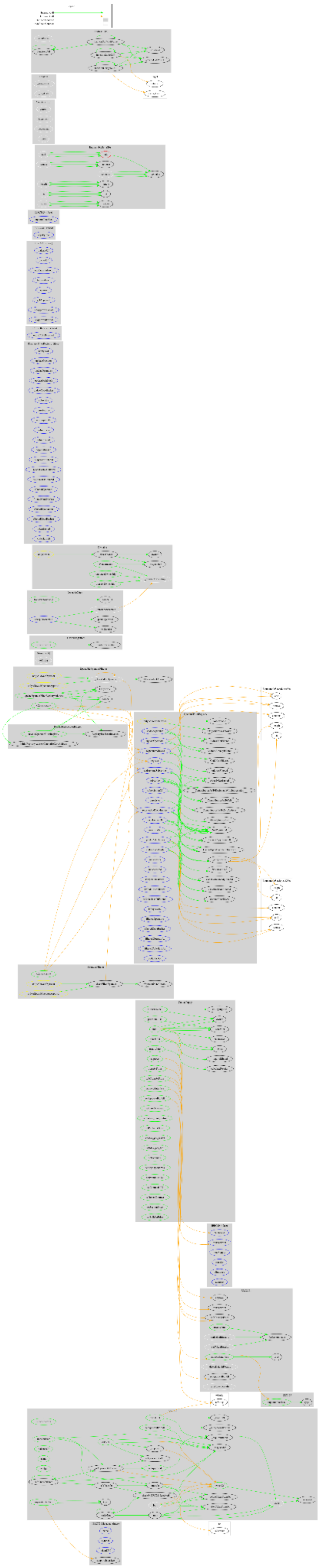
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ZodiacPuppy	Implementation	ERC721, DefaultOperatorFilterer, Ownable, ERC2981		
		Public	✓	ERC721
	preMintNFTs	Public	✓	onlyOwner
	mint	Public	Payable	-
	tokenURI	Public		-
	setApprovalForAll	Public	✓	onlyAllowedOperatorApproval
	approve	Public	✓	onlyAllowedOperatorApproval
	transferFrom	Public	✓	onlyAllowedOperator
	safeTransferFrom	Public	✓	onlyAllowedOperator
	setRoyaltyInfo	Public	✓	onlyOwner
	safeTransferFrom	Public	✓	onlyAllowedOperator
	royaltyInfo	Public		-
	calculateRoyalty	Public		-
	supportsInterface	Public		-
	setBaseExtension	Public	✓	onlyOwner
	setPublic_mint_status	Public	✓	onlyOwner

	setPublicSaleCost	Public	✓	onlyOwner
	setMax_per_wallet	Public	✓	onlyOwner
	setMax_per_txn	Public	✓	onlyOwner
	setDecimals	Public	✓	onlyOwner
	setRoyaltyAddress	Public	✓	onlyOwner
	setPremintwallet	Public	✓	onlyOwner
	setContractURI	Public	✓	onlyOwner
	setTokenContract	Public	✓	onlyOwner
	setSectionPrice	Public	✓	onlyOwner
	setBulkSetPrice	Public	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

ZODIAC PUPPY contract implements a nft mechanism. This audit investigates security issues, business logic concerns and potential improvements. The audit revealed no critical or medium security findings in the contract and only minor / informative findings are reported. The images and metadata associated with the NFTs are unchangeable, ensuring their immutability.

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