Cyberscope

Audit Report

PUMPSPACE

January 2025

Repository https://github.com/bluewhale-logan/pumpspace-contract

Commit 6116e49d2de55c963555a06c884bbbd46111af6b

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/bluewhale-logan/pumpspace-contract
Commit	6116e49d2de55c963555a06c884bbbd46111af6b

Audit Updates

Initial Audit	13 Jan 2025

Source Files

Filename	SHA256
Router.sol	5ed960fb6d87169ed3371b5469aa8a00ddf1fefd7daa4be6d7a8b5c4b215970a
Pair.sol	2607ac74a6478f2d0980e142122e6c5b0f3bef81330e93d5f055f2ed74e780da
Migrator.sol	b190e326d9a8ca887b4afa09838c07d2bf8c20cb1b44915c17f8b632eb2d1886
Factory.sol	2480a11d58af8ab5b8e6673532ea371790eb538cb454ed75653e97eceba633d5
structs/PumpStructs.sol	ec774831e41cc04e590992aac0a3f72e16f648830df426e5058b703bca1a328c

Overview

Pump Space is a decentralized application (dApp) designed to provide users with tools for token creation, liquidity management, and trading. It features an integrated ecosystem of contracts, including a Router, Factory, Pair, Migrator, and MemeToken, each serving a distinct role. These components work together to facilitate token creation, liquidity provision, fee management, and migration to external decentralized exchanges.

Router

The Router acts as the central contract for user interactions within Pump Space. It enables users to create new tokens (`pump` or `pumpWithETH`) and their associated liquidity pools while handling fee calculations and transfers. The Router manages swaps, token buying and selling, and tracks creators and tokens via mappings. It also initiates migration processes when liquidity pools meet specific conditions, ensuring seamless integration with external exchanges.

Factory

The Factory is responsible for creating and managing liquidity pools (Pairs). It initializes pools with default reserve values and assigns them unique addresses, ensuring that no duplicate pools exist for a given token. The Factory maintains a mapping of tokens to their corresponding pools, allowing efficient lookups. It also enforces the Factory-initiated lifecycle of pools, supporting the Router in token and liquidity management.

Pair

The Pair contract represents an individual liquidity pool, maintaining reserves for two tokens and handling swaps between them. It enforces the rules for token trading, tracks trading activity, and calculates fees. Additionally, the Pair supports user tracking and integrates a role-based access control system for secure operations. It emits key events such as `Swap` and `Sync` , ensuring transparency and traceability.

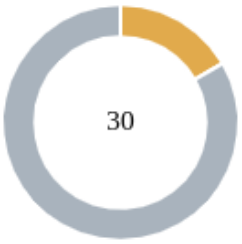
Migrator

The Migrator manages the migration of liquidity from Pump Space pools to external decentralized exchanges. It calculates migration fees, allocates them to creators, referrals, and the designated fee address, and handles token burning to adjust reserves. The Migrator ensures that liquidity transitions are smooth and compliant with the defined fee structure, enabling compatibility with other protocols.

MemeToken

The MemeToken is a customizable ERC-20 token created through the Router. Each token is initialized with metadata such as a name, symbol, description, and social media links. Tokens are minted with a predefined supply and are associated with a creator and an optional referral. MemeToken serves as the core asset for trading and liquidity provisioning within the Pump Space ecosystem.

Findings Breakdown



Critical	0
Medium	5
Minor / Informative	25

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	5	0	0
Minor / Informative	0	25	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	AMF	Automated Migration Failure	Acknowledged
Medium	IFC	Inconsistent Fee Calculation	Acknowledged
Medium	ITH	Incorrect Token Handling	Acknowledged
Medium	MRFM	Manipulable Referral Fee Mechanism	Acknowledged
Medium	MAC	Missing Access Control	Acknowledged
Minor / Informative	MEM	Misleading Error Message	Acknowledged
Minor / Informative	RSV	Redundant Struct Variables	Acknowledged
Minor / Informative	UAC	Unchecked Arithmetic Calculations	Acknowledged
Minor / Informative	USF	Unchecked Swap Fee	Acknowledged
Minor / Informative	CR	Code Repetition	Acknowledged
Minor / Informative	CCR	Contract Centralization Risk	Acknowledged
Minor / Informative	DPR	Duplicate Pair Registration	Acknowledged
Minor / Informative	DTC	Duplicate Token Creation	Acknowledged
Minor / Informative	EPR	Exceeded Purchase Reverts	Acknowledged

●	HV	Hardcoded Values	Acknowledged
●	IDI	Immutable Declaration Improvement	Acknowledged
●	IUDH	Inconsistent User Data Handling	Acknowledged
●	MCM	Misleading Comment Messages	Acknowledged
●	MEE	Missing Events Emission	Acknowledged
●	MSF	Missing Sync Functionality	Acknowledged
●	MU	Modifiers Usage	Acknowledged
●	PBV	Percentage Boundaries Validation	Acknowledged
●	PSU	Potential Subtraction Underflow	Acknowledged
●	RMI	Reserve Misallocation Issue	Acknowledged
●	UPC	Uncontrolled Pair Cloning	Acknowledged
●	UAR	Unutilized Admin Role	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L06	Missing Events Access Control	Acknowledged
●	L16	Validate Variable Setters	Acknowledged
●	L19	Stable Compiler Version	Acknowledged

AMF - Automated Migration Failure

Criticality	Medium
Location	Router.sol#L166
Status	Acknowledged

Description

The `createPoolAndTransfer` function in the router contract calls the swap function of the Pair contract but does not check the `_isTokenSoldout` variable returned from it. This omission can cause an issue when the creator immediately buys the full allocation of tokens, effectively triggering a condition where the tokens are sold out. In such cases, the migration step, which is expected to automate the transition to the next phase, is not performed because the `createPoolAndTransfer` function lacks the necessary logic to handle this scenario. This results in the automated migration process not being executed, potentially disrupting the intended functionality of the system.

```
function createPoolAndTransfer(
        address _token0,
        uint256 _inAmount,
        bool _isETH,
        address sender
    ) internal returns (address pool) {

        ...

        if (swapAmount > 0) {
            (uint256 outAmount, uint256 fee,) =
IPumpPair(pool).swap(sender, swapAmount, true, swapFeeTo);

            _safeTransfer(_token0, sender, outAmount);
            if (fee > 0) _safeTransfer(_token1, swapFeeTo, fee);
        }
        ...
```

Recommendation

It is recommended to check the `_isTokenSoldout` variable returned from the swap function during the execution of the `createPoolAndTransfer` function. If the variable

indicates that the tokens are sold out, the contract should initiate the migration process to ensure seamless automation and alignment with the expected behavior. This enhancement will improve the system's reliability and ensure the correct execution of all phases in the token lifecycle.

Team Update

The team has acknowledged that this is not a security issue.

IFC - Inconsistent Fee Calculation

Criticality	Medium
Location	Router.sol#L82,161
Status	Acknowledged

Description

The `pump` function in the router contract allows users to specify an `initAmount` for token purchases. However, during the `createPoolAndTransfer` process, the function transfers the `initAmount` plus the fee to the contract. This approach increases the total transferred amount beyond the specified `initAmount`, which can result in inconsistencies, particularly if the user's token allowance is set to exactly match the `initAmount`. The increased transfer amount due to the added fee may cause the transaction to fail with insufficient allowance errors, even when the user intended to provide sufficient funds. This inconsistency does not occur in the ETH case, where the fee is correctly deducted from the `initAmount`.

```
    function pump(
         string memory name,
         string memory symbol,
         string memory desc,
         string memory img,
         string[4] memory urls,
         uint256 initAmount,
         address referral
         ) external override returns (address, address) {
         return
initializeTokenAndPool(PumpStructs.InitTokenParams(msg.sender,
referral, name, symbol, desc, img, urls, initAmount, false));
     }

     function createPoolAndTransfer(
         address _token0,
         uint256 _inAmount,
         bool _isETH,
         address sender
     ) internal returns (address pool) {

         address _token1 = _isETH ? WETH : token1;
         uint256 _pumpFee = _isETH ? pumpETHFee : pumpFee;
         uint256 swapAmount = _inAmount;

         ...
         } else {
             _safeTransferFrom(_token1, sender, address(this), _inAmount
+ _pumpFee);
         }
         ...
```

Recommendation

It is recommended to refactor the contract logic to calculate the fee by deducting it from the `initAmount` , similar to how it is handled in the ETH case. This ensures consistency across different scenarios and prevents unexpected allowance errors. The contract should consider only the adjusted `initAmount` and calculate the fee based on the initial amount, ensuring predictable behavior and a smoother user experience.

Team Update

The team has acknowledged that this is not a security issue.

ITH - Incorrect Token Handling

Criticality	Medium
Location	Router.sol#L222 Factory.sol#L38 Pair.sol#L142
Status	Acknowledged

Description

The contract is designed to allow users to purchase tokens using ETH. However, it fails to properly differentiate cases where the `token1` of a trading pair is not ETH but another token. Instead, the contract assumes `token1` behaves as ETH in all scenarios. This incorrect handling can result in unexpected behaviour, including failed transactions or incorrect fee allocations, when `token1` is a different token. Additionally, the contract enforces a restriction where only one pool can be created using a specific `token0`. As a result, it is not possible for a pair to simultaneously have both `token1` and ETH, limiting the flexibility of pool creation. This issue arises from the lack of functionality separation to handle non-ETH tokens as `token1`.

```
function buyToken(address _token, uint256 amountOutMin) external
payable override returns (uint256) {

        ...
        require(pool != address(0), "Pool not found for this token");

        IWETH(WETH).deposit{value: sentValue}();

        (uint256 outAmount, uint256 fee, bool isTokenSoldout) =
IPumpPair(pool).swap(sender, sentValue, true, _swapFeeTo);
        ...
    }
    function createPool(address token0, address token1, address
routerAddress, bool isETH) external onlyRouter returns (address
poolAddress) {
        require(tokenToPool[token0] == address(0), "Pool already exists
for this token");
    ...
    }

 constructor(address _token0, address _token1, uint112 _reserve0,
uint112 _reserve1, address _router, bool _isETH) {
        token0 = _token0;
        token1 = _token1;
        ...
    }

   function swap(
        address account,
        uint256 inAmount,
        bool isBuy,
        address swapFeeTo
    ) external lock nonReentrant onlyRouter returns (uint256 outAmount,
uint256 fee, bool _isTokenSoldout) {
        require(!isTokenSoldout, "Token Sold");

        (uint112 _reserve0, uint112 _reserve1, ) = getReserves();
        uint256 _swapFeeRate = swapFeeTo != address(0) ? swapFeeRate :
0;
        uint256 _totalSoldAmount = totalSoldAmount;

        uint256 balance0 = 0;
        uint256 balance1 = 0;

        if (isBuy) {

            ...

        } else {
            ...
```

```
    }
```

Recommendation

It is recommended to implement a clear separation of functionality to distinguish between `token1` being ETH and `token1` being any other token. The contract should include additional checks and logic to correctly manage non-ETH tokens as `token1`, ensuring proper transaction flow, fee distribution, and output validation. Furthermore, the contract logic should be reviewed to allow the creation of pools that can handle both `token1` and ETH in a flexible manner, or clearly document and justify this limitation.

Team Update

The team has acknowledged that this is not a security issue.

MRFM - Manipulable Referral Fee Mechanism

Criticality	Medium
Location	Router.sol#L82,94,301
Status	Acknowledged

Description

The `transferMigrationFee` function of the `Router` contract allows a fee deducted from the migration process to be transferred to the `referral` address provided by users during the `pump` or `pumpWithETH` functions. However, since the referral address is directly supplied by users, it can be manipulated. Malicious users may provide their own addresses as the referral, enabling them to receive the referral fee intended for legitimate referrals or the `migrationFeeTo` address. This vulnerability could lead to unauthorized fund allocation and exploitation of the referral mechanism.

```
    function pump(
        string memory name,
        string memory symbol,
        string memory desc,
        string memory img,
        string[4] memory urls,
        uint256 initAmount,
        address referral
        ) external override returns (address, address) {
        return
initializeTokenAndPool(PumpStructs.InitTokenParams(msg.sender,
referral, name, symbol, desc, img, urls, initAmount, false));
    }

    function pumpWithETH(
        string memory name,
        string memory symbol,
        string memory desc,
        string memory img,
        string[4] memory urls,
        address referral
    ) external payable override returns (address, address) {
        return
initializeTokenAndPool(PumpStructs.InitTokenParams(msg.sender,
referral, name, symbol, desc, img, urls, msg.value, true));
    }

    function transferMigrationFee(uint256 feeAmount, address _token0,
address _token1, bool isETH) internal {

        address creator = tokenDatas[_token0].creator;
        address referral = tokenDatas[_token0].referral;

        (uint256 creatorAmount, uint256 referralAmount, uint256
migrationFee, address migrationFeeTo) =
migrator.handleMigrationFee(feeAmount, referral, isETH);

        if (isETH) {
            IWETH(_token1).withdraw(feeAmount);
            if (creatorAmount > 0) _safeTransferETH(creator,
creatorAmount);
            if (referralAmount > 0) _safeTransferETH(referral,
referralAmount);
            _safeTransferETH(migrationFeeTo, migrationFee);
        } else {
            if (creatorAmount > 0) _safeTransfer(_token1, creator,
creatorAmount);
            if (referralAmount > 0) _safeTransfer(_token1, referral,
referralAmount);
            _safeTransfer(_token1, migrationFeeTo, migrationFee);
```

```
        }
    }
```

Recommendation

It is recommended to enhance the referral validation process to prevent misuse. The contract could verify if the referral address is already part of the system, such as by checking whether the address has previously participated in specific activities (e.g., token creation or active buy amount). Alternatively, the fee allocation logic could be adjusted to base the referral fee on the `creator`'s contribution, or the system could enforce stricter rules for referral eligibility. These measures will help ensure that referral fees are appropriately distributed and protect against abuse.

Team Update

The team has acknowledged that this is not a security issue.

MAC - Missing Access Control

Criticality	Medium
Location	Router.sol#L56 Migrator.sol#L23 Factory.sol#L24
Status	Acknowledged

Description

The `initialize` functions can be frontrun during deployment, allowing administrative roles to be transferred to third parties not associated with the team. Such third parties would gain access to all the functions of the system.

```
    function initialization(
        address _factory, address _migrator, address _token1, address
_wETH, address _pumpFeeTo, address _swapFeeTo
    ) public initializer {
        ...
    }

    function initialization() public initializer {
        __Ownable_init();

        defaultReserve0 = 1_073_000_000 * 10 ** 18;
        defaultReserve1 = 4000 * 10 ** 18;
        defaultReserve1ByETH = 120 * 10 ** 18;
    }

    function initialization(address _feeTo) public initializer {
        __Ownable_init();

        feeTo = _feeTo;

        migrationFee = 7;

        creatorFee = 40;
        referralFee = 10;

        creatorFeeETH = 20;
        referralFeeETH = 5;
    }
```

Recommendation

The team is advised to implement proper access controls to ensure that only authorized team members can call these functions.

Team Update

The team has acknowledged that this is not a security issue.

MEM - Misleading Error Message

Criticality	Minor / Informative
Location	Pair.sol#L183
Status	Acknowledged

Description

The contract contains a `require` statement with the error message `"Not enough ETH"` that is triggered when `amountOutWithFee` exceeds `_reserve1`. However, `_reserve1` does not necessarily represent ETH, as it can correspond to any token depending on the pair's configuration. This inconsistency between the error message and the actual functionality can mislead developers and users, potentially causing confusion when diagnosing and resolving transaction failures. Misleading error messages reduce the contract's readability and increase the likelihood of misunderstanding its behavior.

```
require(amountOutWithFee <= _reserve1, "Not enough ETH");
```

Recommendation

It is recommended to update the error message to accurately reflect the condition being checked. For example, a message like `"Insufficient reserves for token output"` would more accurately describe the scenario and apply universally, regardless of the token type in `_reserve1`. Ensuring clear and context-appropriate error messages improves the contract's usability and aids in efficient debugging.

Team Update

The team has acknowledged that this is not a security issue.

RSV - Redundant Struct Variables

Criticality	Minor / Informative
Location	PumpStructs.sol#L6
Status	Acknowledged

Description

The `InitTokenParams` and `TokenData` structs contain several overlapping variables, such as `creator`, `referral`, `description`, `image`, and `isETH`. This redundancy increases the potential for inconsistencies between different parts of the contract where these structs are used or updated. Additionally, duplicating variables across structs unnecessarily increases storage and computational costs, particularly when these structs are passed or stored in mappings.

```
struct InitTokenParams {
        address creator;
        address referral;
        string name;
        string symbol;
        string description;
        string image;
        string[4] urls;
        uint256 inAmount;
        bool isETH;
    }


    struct TokenData {
        address creator;
        address referral;
        address token;
        address pair;
        string description;
        string image;
        string twitter;
        string telegram;
        string youtube;
        string website;
        bool trading;
        bool tradingOnUniswap;
        bool isETH;
    }

    struct Creator {
        address user;
        address[] tokens;
    }

    struct Holder {
        address user;
        uint256 swappedAmount;
    }
```

Recommendation

It is recommended to refactor the structs to minimize redundancy. Shared variables should be extracted into a base or shared struct that both `InitTokenParams` and `TokenData` can reference. For instance, a separate struct for common metadata (e.g., `BasicTokenInfo`) could help reduce duplication. This approach will improve the

maintainability of the code, reduce storage costs, and prevent potential mismatches or errors caused by redundant variables.

Team Update

The team has acknowledged that this is not a security issue.

UAC - Unchecked Arithmetic Calculations

Criticality	Minor / Informative
Location	Pair.sol#L129
Status	Acknowledged

Description

The contract is implemented with an `unchecked` block in the code handling cumulative price updates. While the `unchecked` keyword can optimise gas costs by bypassing Solidity's default overflow and underflow checks, it introduces a significant risk. If variables such as `blockTimestamp`, `blockTimestampLast`, `_reserve0`, or `_reserve1` are manipulated or unexpected values are encountered, overflow or underflow issues could occur. This could lead to incorrect price calculations, impacting trading dynamics and reserve management within the contract.

```
        unchecked {
            uint32 timeElapsed = blockTimestamp - blockTimestampLast;
            if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
                price0CumulativeLast +=
uint256(CustomUQ112x112.encode(_reserve1).divide(_reserve0)) *
timeElapsed;
                price1CumulativeLast +=
uint256(CustomUQ112x112.encode(_reserve0).divide(_reserve1)) *
timeElapsed;
            }
        }
```

Recommendation

It is recommended to carefully review and validate all inputs to the `unchecked` block to ensure they remain within safe boundaries. Additionally, consider implementing explicit range checks or alternative safety mechanisms to mitigate overflow or underflow risks. While `unchecked` can improve efficiency, its use should be limited to scenarios where there is absolute certainty of input integrity.

Team Update

The team has acknowledged that this is not a security issue.

USF - Unchecked Swap Fee

Criticality	Minor / Informative
Location	Pair.sol#L228
Status	Acknowledged

Description

The contract is implemented with a `setSwapFeeRate` function, which allows the owner to modify the `swapFeeRate` variable without any constraints or validation. Since this variable directly impacts the calculation of swap fees, any unintended or malicious changes to it could distort token prices and potentially destabilise the reserve balance. This creates a significant risk, as an excessively high fee rate could disincentivise swaps or cause financial losses to users, while an unreasonably low fee rate might result in insufficient fees to maintain reserve sustainability.

```
    function setSwapFeeRate(uint256 _swapFeeRate) public onlyOwner {
        swapFeeRate = _swapFeeRate;
    }
```

Recommendation

It is recommended to implement robust checks on the `swapFeeRate` variable to ensure that updates remain within a safe and predefined range. Additionally, consider introducing a governance mechanism or multi-signature approval process to limit unilateral modifications by a single actor.

Team Update

The team has acknowledged that this is not a security issue.

CR - Code Repetition

Criticality	Minor / Informative
Location	Router.sol#L189,222,247
Status	Acknowledged

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
    function swapToken(address _token, uint256 amountIn, uint256
amountOutMin, bool isBuy) external override returns (uint256) {
        ...

        return outAmount;
    }

    function buyToken(address _token, uint256 amountOutMin) external
payable override returns (uint256) {

        ...

        return outAmount;
    }

    function sellToken(address _token, uint256 amountIn, uint256
amountOutMin) external override returns (uint256) {
        ...

        return outAmount;
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the

contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

Team Update

The team has acknowledged that this is not a security issue.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Router.sol#L320 Factory.sol#L67 Pair.sol#L228 Migrator.sol#L80
Status	Acknowledged

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.

```
    function setSwapFeeTo(address _swapFeeTo) external onlyOwner {
        swapFeeTo = _swapFeeTo;
    }

    function setPumpFeeTo(address _pumpFeeTo) external onlyOwner {
        pumpFeeTo = _pumpFeeTo;
    }

    function setFactory(address _factory) external onlyOwner {
        factory = IPumpFactory(_factory);
    }

    function setRouter(address _router) public onlyOwner {
        router = _router;
    }

    function addPairRouter(address _pair, address _router) public
onlyOwner {
        Pair(_pair).addRouter(_router);
    }

    function removePairRouter(address _pair, address _router) public
onlyOwner {
        Pair(_pair).removeRouter(_router);
    }

    function setCreatePairReserve0(uint112 _defaultReserve0) external
onlyOwner {
        defaultReserve0 = _defaultReserve0;
    }

    function setCreatePairReserve1(uint112 _defaultReserve1) external
onlyOwner {
        defaultReserve1 = _defaultReserve1;
    }

    function setCreatePairReserve1ByETH(uint112 _defaultReserve1ByETH)
external onlyOwner {
        defaultReserve1ByETH = _defaultReserve1ByETH;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function setSwapFeeRate(uint256 _swapFeeRate) public onlyOwner {
        swapFeeRate = _swapFeeRate;
    }

    function setDexRouter(address _dexRouter) external onlyOwner {
```

```
        dexRouter = _dexRouter;
    }

    function setMigrationFee(uint256 _migrationFee) external onlyOwner
{
        migrationFee = _migrationFee;
    }

    function setCreatorFee(uint256 _creatorFee) external onlyOwner {
        creatorFee = _creatorFee;
    }

    function setReferralFee(uint256 _referralFee) external onlyOwner {
        referralFee = _referralFee;
    }

    function setCreatorFeeETH(uint256 _creatorFeeETH) external
onlyOwner {
        creatorFeeETH = _creatorFeeETH;
    }

    function setReferralFeeETH(uint256 _referralFeeETH) external
onlyOwner {
        referralFeeETH = _referralFeeETH;
    }
```

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.

Team Update

The team has acknowledged that this is not a security issue.

DPR - Duplicate Pair Registration

Criticality	Minor / Informative
Location	Factory.sol#L38
Status	Acknowledged

Description

The contract is designed to allow the creation of liquidity pools for token pairs. However, it only performs a check to ensure that a pool for the `token0` address does not already exist. It does not verify whether a pair for both `token1-token0` and `token0-token1` already exists, as is the standard practice in most implementations. This omission can result in both `token1-token0` and `token0-token1` pairs being registered simultaneously, causing confusion and potential inconsistencies in liquidity allocation and trading.

```
   function createPool(address token0, address token1, address
routerAddress, bool isETH) external onlyRouter returns (address
poolAddress) {
        require(tokenToPool[token0] == address(0), "Pool already exists
for this token");
    ...
    }
```

Recommendation

It is recommended to implement a comprehensive validation mechanism that ensures a pair is registered only once, regardless of the order of tokens (`token0` and `token1`). The contract should verify the existence of both `token1-token0` and `token0-token1` pairs before proceeding with the creation of a new pool to prevent duplication and maintain clarity in the liquidity pool system.

Team Update

The team has acknowledged that this is not a security issue.

DTC - Duplicate Token Creation

Criticality	Minor / Informative
Location	tokens/MemeToken.sol#L19
Status	Acknowledged

Description

The `MemeToken` contract is designed to create new meme tokens based on specific characteristics provided during deployment. However, it does not ensure that tokens with the same characteristics, such as `description`, `image`, or other properties, are distinguished from one another. As a result, two or more tokens with identical characteristics but different deployment addresses may coexist. This could lead to confusion among users or token holders, as the tokens might appear identical but functionally differ based on their contract addresses. This issue undermines the uniqueness and integrity of each token instance created by the contract.

```
    constructor(
        string memory _name,
        string memory _symbol,
        PumpStructs.InitTokenParams memory params
    ) ERC20(_name, _symbol) {

        creator = params.creator;
        referral = params.referral;
        description = params.description;
        image = params.image;
        twitter = params.urls[0];
        telegram = params.urls[1];
        youtube = params.urls[2];
        website = params.urls[3];
        isETH = params.isETH;

        _mint(msg.sender, 1_000_000_000 * 10 ** 18);
    }
```

Recommendation

It is recommended to implement a mechanism to validate the uniqueness of the characteristics provided during token creation. Consider including a registry or mapping to track existing tokens based on their defining characteristics, ensuring that duplicate tokens cannot be deployed. Additionally, you may implement a hashing or signature method to enforce the uniqueness of each token's attributes. This approach will maintain the intended distinction and prevent potential misuse or confusion among users.

Team Update

The team has acknowledged that this is not a security issue.

EPR - Exceeded Purchase Reverts

Criticality	Minor / Informative
Location	Pair.sol#L169
Status	Acknowledged

Description

The contract is designed to enforce a limit on the total tokens that can be sold (`_totalSoldAmount`) by reverting transactions where the desired purchase amount (`outAmount`) would cause this limit to be exceeded. While this ensures the cap is not breached, it results in a poor user experience for buyers attempting to purchase tokens near the limit, as their transactions revert entirely instead of partially fulfilling the request for the remaining available tokens.

```
   function swap(
        address account,
        uint256 inAmount,
        bool isBuy,
        address swapFeeTo
    ) external lock nonReentrant onlyRouter returns (uint256 outAmount,
uint256 fee, bool _isTokenSoldout) {
       ...

        if (isBuy) {
            ...

            require(_totalSoldAmount < 820_000_000 * 10 ** 18,
"MAX_820_000_000 reached");

            if (_totalSoldAmount >= 800_000_000 * 10 ** 18) {
                isTokenSoldout = true;
                _isTokenSoldout = true;
            }
```

Recommendation

It is recommended to refactor the code to calculate the remaining tokens available for purchase when the limit is approached. The contract should allocate these remaining tokens to the buyer, adjust the `_totalSoldAmount` accordingly, and set the

`isTokenSoldout` flag to `true` once the limit is reached. This approach would enhance the user experience, ensure the efficient and complete allocation of tokens, and provide a clear indication of the token's sold-out status.

Team Update

The team has acknowledged that this is not a security issue.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	Router.sol#L278 Factory.sol#L24 Pair.sol#L102,169 Migrator.sol#L49
Status	Acknowledged

Description

The contract contains multiple instances where numeric values are directly hardcoded into the code logic rather than being assigned to constant variables with descriptive names. Hardcoding such values can lead to several issues, including reduced code readability, increased risk of errors during updates or maintenance, and difficulty in consistently managing values throughout the contract. Hardcoded values can obscure the intent behind the numbers, making it challenging for developers to modify or for users to understand the contract effectively.

```
if( _migrationDexRouter != address(0) && 800_000_000 * 10 ** 18 <=
pair.totalSoldAmount() ) {
```

```
    function initialization() public initializer {
        __Ownable_init();

        defaultReserve0 = 1_073_000_000 * 10 ** 18;
        defaultReserve1 = 4000 * 10 ** 18;
        defaultReserve1ByETH = 120 * 10 ** 18;
    }
```

```
uint256 _minReserveAfterSale = 253_000_000 * 10 ** 18;
...
    require(_totalSoldAmount < 820_000_000 * 10 ** 18, "MAX_820_000_000
reached");
if (_totalSoldAmount >= 800_000_000 * 10 ** 18) {
    isTokenSoldout = true;
    _isTokenSoldout = true;
}
```

```
uint256 defaultReserve0 = 73_000_000 * 10 ** 18;
```

Recommendation

It is recommended to replace hardcoded numeric values with variables that have meaningful names. This practice improves code readability and maintainability by clearly indicating the purpose of each value, reducing the likelihood of errors during future modifications. Additionally, consider using constant variables which provide a reliable way to centralize and manage values, improving gas optimization throughout the contract.

Team Update

The team has acknowledged that this is not a security issue.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Pair.sol#L57,58,62,63,66,69 tokens/MemeToken.sol#L25,26,33
Status	Acknowledged

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
token0
token1
factory
router
isETH
reserve1Init
creator
referral
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

Team Update

The team has acknowledged that this is not a security issue.

IUDH - Inconsistent User Data Handling

Criticality	Minor / Informative
Location	Pair.sol#L201
Status	Acknowledged

Description

The `Pair` contract include the `updateUserData` function, which adds a user to the `users` array during a "buy" operation if the user does not already exist. However, it does not remove the user when their `userSwappedAmount` becomes zero after a "sell" operation. This results in the `users` array containing entries for accounts with zero balances, unnecessarily bloating the array with irrelevant data. This inconsistency in handling user data can lead to inefficient storage usage and increased gas costs for future operations involving the `users` array.

```
    function updateUserData(address account, uint256 outAmount, bool
isBuy) private {

        if (isBuy) {
            if (!userExists[account]) {
                users.push(account);
                userExists[account] = true;
            }
            userSwappedAmount[account] += outAmount;

        } else {
            userSwappedAmount[account] -= outAmount;
        }
    }
```

Recommendation

It is recommended to implement logic to remove users from the `users` array when their `userSwappedAmount` reaches zero. This can help maintain a clean and efficient data structure. Alternatively, if maintaining such entries is required for historical or tracking purposes, consider documenting this behavior to ensure clarity and consistency in the contract's functionality.

Team Update

The team has acknowledged that this is not a security issue.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	Router.sol#L211
Status	Acknowledged

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
if (outAmount < amountOutMin)    //  슬리피지 계산을 위해
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

Team Update

The team has acknowledged that this is not a security issue.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Pair.sol#L228 Migrator.sol#L84
Status	Acknowledged

Description

The contracts perform 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 setSwapFeeRate(uint256 _swapFeeRate) public onlyOwner {
    swapFeeRate = _swapFeeRate;
}
```

```
   function setMigrationFee(uint256 _migrationFee) external onlyOwner {
        migrationFee = _migrationFee;
    }

    function setCreatorFee(uint256 _creatorFee) external onlyOwner {
        creatorFee = _creatorFee;
    }

    function setReferralFee(uint256 _referralFee) external onlyOwner {
        referralFee = _referralFee;
    }

    function setCreatorFeeETH(uint256 _creatorFeeETH) external
onlyOwner {
        creatorFeeETH = _creatorFeeETH;
    }

    function setReferralFeeETH(uint256 _referralFeeETH) external
onlyOwner {
        referralFeeETH = _referralFeeETH;
    }
```

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.

Team Update

The team has acknowledged that this is not a security issue.

MSF - Missing Sync Functionality

Criticality	Minor / Informative
Location	Pair.sol
Status	Acknowledged

Description

The `Pair` contract does not include a `.sync` function, which is typically used in Automated Market Maker (AMM) designs to ensure the stored reserves (`reserve0` and `reserve1`) are consistent with the actual token balances held by the contract. The absence of this functionality may lead to issues if tokens are accidentally or intentionally transferred directly to the contract without going through its controlled functions. Such discrepancies can result in inaccurate reserve values, leading to calculation errors during swaps or other reserve-dependent operations. This can compromise the reliability of the contract and potentially cause unexpected behavior.

Recommendation

It is recommended to consider including a `.sync` function in the Pair contract to manually synchronize the stored reserves with the actual token balances in the contract. This function can act as a safeguard to restore reserve consistency in scenarios where external transfers or unforeseen events lead to discrepancies. Implementing this functionality ensures that the contract operates reliably and maintains accurate reserve values at all times.

Team Update

The team has acknowledged that this is not a security issue.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	Router.sol#L193,236,253,259
Status	Acknowledged

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
address pool = factory.getPoolByToken(_token);
require(pool != address(0), "Pool not found for this token pair");
...
if (outAmount < amountOutMin)
    revert InsufficientAmount("Router: INSUFFICIENT_OUTPUT_AMOUNT");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

Team Update

The team has acknowledged that this is not a security issue.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	Pair.sol#L228 Migrator.sol#L84
Status	Acknowledged

Description

The contracts utilize variables for percentage-based calculations that are required for its operations. These variables are involved in multiplication and division operations to determine proportions related to the contract's logic. If such variables are set to values beyond their logical or intended maximum limits, it could result in incorrect calculations. This misconfiguration has the potential to cause unintended behavior or financial discrepancies, affecting the contract's integrity and the accuracy of its calculations.

```
function setSwapFeeRate(uint256 _swapFeeRate) public onlyOwner {
    swapFeeRate = _swapFeeRate;
}
```

```
   function setMigrationFee(uint256 _migrationFee) external onlyOwner {
        migrationFee = _migrationFee;
    }

    function setCreatorFee(uint256 _creatorFee) external onlyOwner {
        creatorFee = _creatorFee;
    }

    function setReferralFee(uint256 _referralFee) external onlyOwner {
        referralFee = _referralFee;
    }

    function setCreatorFeeETH(uint256 _creatorFeeETH) external
onlyOwner {
        creatorFeeETH = _creatorFeeETH;
    }

    function setReferralFeeETH(uint256 _referralFeeETH) external
onlyOwner {
        referralFeeETH = _referralFeeETH;
```

```
    }
```

Recommendation

To mitigate risks associated with boundary violations, it is important to implement validation checks for variables used in percentage-based calculations. Ensure that these variables do not exceed their maximum logical values. This can be accomplished by incorporating `require` statements or similar validation mechanisms whenever such variables are assigned or modified. These safeguards will enforce correct operational boundaries, preserving the contract's intended functionality and preventing computational errors.

Team Update

The team has acknowledged that this is not a security issue.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	Migrator.sol#L41
Status	Acknowledged

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

The `migrationFeeCalc` function computes various reserve values, including `burnAmountReserve0`, which is derived from `_reserve0`, `defaultReserve0`, and `correspondingReserve0`. However, the calculation does not verify whether the resulting `burnAmountReserve0` is a positive number. In scenarios where `_reserve0` is insufficient to cover `defaultReserve0` and `correspondingReserve0`, an underflow could occur, leading to erroneous values and potentially breaking downstream logic or reverting transactions.

```
   function migrationFeeCalc(IPumpPair pair) public view returns
(uint256, uint256, uint256, uint256) {
        (uint112 _reserve0, uint112 _reserve1, ) = pair.getReserves();
        uint112 initReserve1 =  pair.reserve1Init();
        uint256 calcReserve1 = _reserve1 - initReserve1;

        uint256 feeAmount = (calcReserve1 * migrationFee) / 100;
        uint256 remainingReserve1 = calcReserve1 - feeAmount;
        uint256 correspondingReserve0 = (remainingReserve1 * _reserve0)
/ _reserve1;
        uint256 defaultReserve0 = 73_000_000 * 10 ** 18;
        uint256 burnAmountReserve0 = _reserve0 - defaultReserve0 -
correspondingReserve0;

        return (burnAmountReserve0, correspondingReserve0,
remainingReserve1, feeAmount);
    }
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

It is recommended to add a check to ensure that `burnAmountReserve0` is non-negative before returning the value. If the value is negative, the function should handle this appropriately, such as by setting it to zero or reverting the transaction with a descriptive error message. This safeguard will prevent potential underflows and ensure the function operates as intended, even in edge cases.

Team Update

The team has acknowledged that this is not a security issue.

RMI - Reserve Misallocation Issue

Criticality	Minor / Informative
Status	Acknowledged

Description

The contract is designed to initialise reserves (`reserve0` and `reserve1`) during its deployment. However, it does so without handling the actual transfer of tokens to the pair contract. This creates a discrepancy between the stated reserves and the actual token balances of the contract, potentially leading to misrepresentation of liquidity and manipulation of the pool's functionality. This issue can result in vulnerabilities that exploit the disparity between the reserves and the actual token holdings.

```
constructor(address _token0, address _token1, uint112 _reserve0,
uint112 _reserve1, address _router, bool _isETH) {
        token0 = _token0;
        token1 = _token1;
        price0CumulativeLast = 0;
        price1CumulativeLast = 0;
        blockTimestampLast = uint32(block.timestamp);
        factory = msg.sender;
        router = _router;
        swapFeeRate = 10;

        isETH = _isETH;
        reserve0 = _reserve0;
        reserve1 = _reserve1;
        reserve1Init = _reserve1;

...
    }
```

Recommendation

It is recommended to ensure that reserve initialisation aligns with the actual transfer of tokens to the pair contract. Token balances should be synchronised with the declared reserves to maintain consistency, prevent exploitation, and ensure the integrity of the liquidity pool.

Team Update

The team has acknowledged that this is not a security issue.

UPC - Uncontrolled Pair Cloning

Criticality	Minor / Informative
Location	Factory.sol#L46
Status	Acknowledged

Description

The contract is designed to create a new pair using direct instantiation (`new Pair(...)`), but it does not include mechanisms to restrict or control the cloning of the `Pair` contract. This means that the `Pair` logic can be replicated by anyone outside the intended factory or deployment process. As a result, unauthorized clones of the `Pair` contract can be deployed, mimicking legitimate pairs while potentially introducing malicious behaviour or inconsistencies. This lack of control over pair creation could lead to confusion, exploitation, and reduced trust in the protocol.

```
Pair newPool = new Pair(token0, token1, _reserve0, _reserve1,
routerAddress, isETH);
```

Recommendation

It is recommended to implement strict controls within the `Pair` contract to ensure it can only be deployed by the intended factory or authorized addresses. This can be achieved by requiring the factory address in the constructor and validating it during deployment. Additionally, mechanisms should be added to verify and recognize only authorized pairs within the protocol to prevent unauthorized or malicious clones.

Team Update

The team has acknowledged that this is not a security issue.

UAR - Unutilized Admin Role

Criticality	Minor / Informative
Location	Pair.sol#L74
Status	Acknowledged

Description

The pair contract is found to declare the `DEFAULT_ADMIN_ROLE` and assign it to the factory address (via `msg.sender`) during deployment. However, throughout the implementation, this role is not utilized in any function or logic within the contract. Instead, the `msg.sender` (the router contract) is responsible for managing operations. As a result, the inclusion of the `DEFAULT_ADMIN_ROLE` serves no functional purpose and may lead to confusion or unnecessary complexity in understanding the contract's design and intended role hierarchy.

```
    constructor(address _token0, address _token1, uint112 _reserve0,
uint112 _reserve1, address _router, bool _isETH) {
        ...
        factory = msg.sender;
        router = _router;
        ...

        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        grantRole(ROUTER_ROLE, _router);
    }
```

Recommendation

It is recommended to consider the intended functionality of the `DEFAULT_ADMIN_ROLE`. If the role was intended to control specific administrative functions, the contract should be updated to enforce its use in relevant operations. Otherwise, if it was not meant to serve any functional purpose, it can be safely removed to simplify the contract's logic and reduce unnecessary gas costs associated with role management.

Team Update

The team has acknowledged that this is not a security issue.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Router.sol#L44,57,145,146,147,179,189,222,247,269,301,320,324,328,332,340 Pair.sol#L215,228 Migrator.sol#L23,76,80,84,88,92,96,100 Factory.sol#L67,71,75,79,83,87
Status	Acknowledged

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.

```
address public WETH
address _token1
address _wETH
address _migrator
address _swapFeeTo
address _pumpFeeTo
address _factory
address _token0
uint256 _inAmount
bool _isETH
address _token
uint256 amountOu
uint256 amountIn
(bool isETH) publ

...
```

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/stable/style-guide.html#naming-conventions.

Team Update

The team has acknowledged that this is not a security issue.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	Factory.sol#L68
Status	Acknowledged

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.

```
router = _router
```

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.

Team Update

The team has acknowledged that this is not a security issue.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Router.sol#L64,65,67,68,321,325 Pair.sol#L57,58,63 Migrator.sol#L26,77,81 Factory.sol#L68
Status	Acknowledged

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.

```
token1 = _token1
WETH = _wETH
swapFeeTo = _swapFeeTo
pumpFeeTo = _pumpFeeTo
...
token0 = _token0
router = _router
feeTo = _feeTo
dexRouter = _dexRouter
```

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.

Team Update

The team has acknowledged that this is not a security issue.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	tokens/MemeToken.sol#L2 Router.sol#L2 Pair.sol#L2 Migrator.sol#L2 interfaces/IPumpRouter.sol#L2 interfaces/IPumpPair.sol#L2 interfaces/IPumpMigrator.sol#L2 interfaces/IPumpFactory.sol#L2 Factory.sol#L2
Status	Acknowledged

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 ^0.8.0;
```

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.

Team Update

The team has acknowledged that this is not a security issue.

Functions Analysis

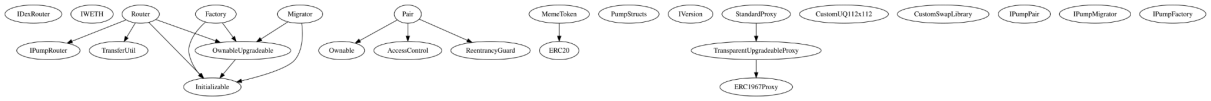
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Router	Implementation	IPumpRouter, Initializable, OwnableUpgradeable, TransferUtil		
	initialization	Public	✓	initializer
		External	Payable	-
	version	External		-
	pump	External	✓	-
	pumpWithETH	External	Payable	-
	initializeTokenAndPool	Internal	✓	
	createPoolAndTransfer	Internal	✓	
	getAmountOut	External		-
	swapToken	External	✓	-
	buyToken	External	Payable	-
	sellToken	External	✓	-
	runDexMigration	Public	✓	onlyOwner
	_migration	Internal	✓	
	transferMigrationFee	Internal	✓	
	setSwapFeeTo	External	✓	onlyOwner
	setPumpFeeTo	External	✓	onlyOwner
	setFactory	External	✓	onlyOwner

	getToken	Public		-
	getTokensByUser	Public		-
	setMigrator	External	✓	onlyOwner
Pair	Implementation	Ownable, AccessControl, ReentrancyGuard		
		Public	✓	-
	addRouter	Public	✓	onlyOwner
	removeRouter	Public	✓	onlyOwner
	getReserves	Public		-
	getBuyAmountOut	Public		-
	getSellAmountOut	Public		-
	_update	Private	✓	
	swap	External	✓	lock nonReentrant onlyRouter
	updateUserData	Private	✓	
	calculateFee	Internal		
	setSwapFeeRate	Public	✓	onlyOwner
	getHolders	External		-
Migrator	Implementation	Initializable, OwnableUpgradeable		
	initialization	Public	✓	initializer
	version	External		-

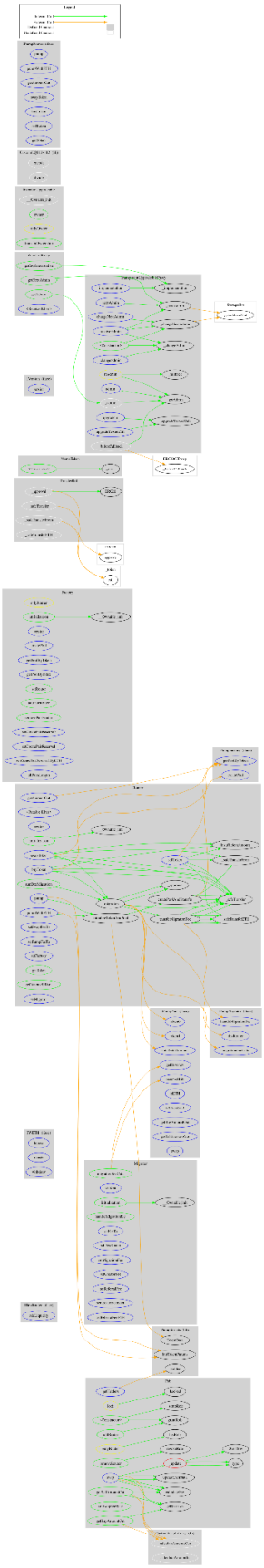
	migrationFeeCalc	Public		-
	handleMigrationFee	Public		-
	setFeeTo	External	✓	onlyOwner
	setDexRouter	External	✓	onlyOwner
	setMigrationFee	External	✓	onlyOwner
	setCreatorFee	External	✓	onlyOwner
	setReferralFee	External	✓	onlyOwner
	setCreatorFeeETH	External	✓	onlyOwner
	setReferralFeeETH	External	✓	onlyOwner
Factory	Implementation	Initializable, OwnableUpgradeable		
	initialization	Public	✓	initializer
	version	External		-
	createPool	External	✓	onlyRouter
	getPoolByToken	External		-
	getPoolByIndex	External		-
	setRouter	Public	✓	onlyOwner
	addPairRouter	Public	✓	onlyOwner
	removePairRouter	Public	✓	onlyOwner
	setCreatePairReserve0	External	✓	onlyOwner
	setCreatePairReserve1	External	✓	onlyOwner
	setCreatePairReserve1ByETH	External	✓	onlyOwner
	allPairsLength	External		-

MemeToken	Implementation	ERC20		
		Public	✓	ERC20

Inheritance Graph



Flow Graph



Summary

Pump Space contract implements a decentralized token creation and liquidity management mechanism. This audit investigates security issues, business logic concerns, and potential improvements to ensure the integrity of token creation, liquidity handling, migration processes, and fee allocation while maintaining a seamless user experience. The team has acknowledged the findings.

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

cyberscope.io