Cyberscope

Audit Report

Papi Coin

May 2023

Network	BSC
Address	0xb381c095676f24ece06591b5cb9b96eec939435e
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Passed
	OCTD	Transfers Contract's Tokens	Passed
	OTUT	Transfers User's Tokens	Passed
	ELFM	Exceeds Fees Limit	Passed
	ULTW	Transfers Liquidity to Team Wallet	Passed
	MT	Mints Tokens	Passed
	BT	Burns Tokens	Passed
	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RSW	Redundant Storage Writes	Unresolved
●	AFI	Accumulated Fees Inconsistency	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	CR	Code Repetition	Unresolved
●	MAP	Misleading Admin Permission	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Papi
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	99999 runs
Explorer	https://bscscan.com/address/0xb381c095676f24ece06591b5cb9b96eec939435e
Address	0xb381c095676f24ece06591b5cb9b96eec939435e
Network	BSC
Symbol	PAPI
Decimals	18
Total Supply	1,000,000,000,000

Audit Updates

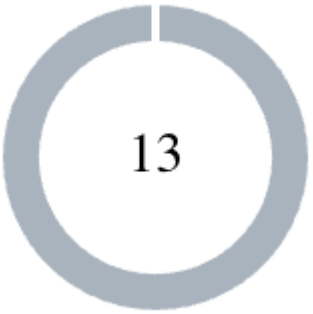
Initial Audit	27 May 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/AccessControl.sol	0ab66c9c0b45fca5efad935058e889bd5bb5599eb95b0d17ec924f64ebcaf38f
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/Strings.sol	8597c62818dcbc6cf85c21179b90b714fb4f70a4347ca2eed23e88c87b08b8a1
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
contracts/interfaces/IPapiDistribution.sol	3de6137c797b4a40b379508ae7d5ad1f75cd381e320ed636c58d9c1b0f1b35d2

contracts/Papi.sol	3354af45c3207edd5d59ec7bca5bb818ffab8313bd6e8c58e18f5e3c9cb4cd11
contracts/PapiDistribution.sol	70efbb3e7c9c565bbca78fa72be274bf088f2cd98cc3d7b82859cf8e18b55b05

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/Papi.sol#L176,182,527
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the excluded from Fee and Swap status of an account even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function setExcludedFromFee(address _address, bool _isExcludedFromFee) public
onlyRole(DEFAULT_ADMIN_ROLE) {
    excludedFromFee[_address] = _isExcludedFromFee;

    emit ExcludedFromFee(_address, _isExcludedFromFee);
}

function setExcludedFromSwap(address _address, bool _isExcludedFromSwap)
public onlyRole(DEFAULT_ADMIN_ROLE) {
    excludedFromSwap[_address] = _isExcludedFromSwap;

    emit ExcludedFromSwap(_address, _isExcludedFromSwap);
}
```

The contract updates the router address and the toke1 address even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function _setRouterAndPair(IUniswapV2Router02 _router, address _token1)
internal {
    require(_token1 != address(0), "zero token1 address");

    address _pair =
IUniswapV2Factory(_router.factory()).getPair(address(this), _token1);

    if (_pair == address(0)) {
        _pair = IUniswapV2Factory(_router.factory()).createPair(address(this),
_token1);
    }

    router = _router;
    token1 = _token1;
    pair = _pair;
    isLpToken[pair] = true;
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

AFI - Accumulated Fees Inconsistency

Criticality	Minor / Informative
Location	contracts/Papi.sol#L222,311,337
Status	Unresolved

Description

The contract resets the accumulated fees variables without performing a distribution of the accumulated rewards. This creates an inconsistency between the actual accumulated tokens from the fees, as the reset effectively clears the stored values without properly accounting for their distribution.

```
function resetRewardsAmount() external onlyRole(DEFAULT_ADMIN_ROLE) {
    rewardSellAmount = 0;
    rewardBuyAmount = 0;

    emit RewardsAmountReseted();
}

function resetLiquidityFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
    liquidityFeeAmount = 0;

    emit LiquidityFeeReseted();
}

function resetSwapFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
    swapFeeAmount = 0;

    emit SwapFeeReseted();
}
```

Recommendation

It is recommended to perform proper distribution of the accumulated rewards before resetting the accumulated variables. This ensures that users receive their deserved rewards based on their interactions with the contract.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	contracts/Papi.sol#L391
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The contract incorporates the capability to distribute burn fees among multiple addresses. On the contrary, the burn mechanism facilitates the transfer of tokens to a dead address for the purpose of burning them.

```
uint256 public burnFeeBuyRate;
uint256 public burnFeeSellRate;
uint256 public burnFeeTransferRate;
address[] public burnFeeReceivers;
uint256[] public burnFeeReceiversRate;

for (uint256 i = 0; i < burnFeeReceivers.length; i++) {
    _transferAmount(_from, burnFeeReceivers[i], _calcFee(burnFeeRes,
burnFeeReceiversRate[i]));
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/Papi.sol#L192,200,275,283,295,303,321,329,393,461,470,478
Status	Unresolved

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.

```
require(_rewardSwapReceivers.length == _rewardSwapReceiversRate.length,
"size");

uint256 _totalRate = 0;
for (uint256 _i = 0; _i < _rewardSwapReceiversRate.length; _i++) {
    _totalRate += _rewardSwapReceiversRate[_i];
}
require(_totalRate == 10000, "rate");
```

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.

MAP - Misleading Admin Permission

Criticality	Minor / Informative
Location	contracts/Papi.sol#L22
Status	Unresolved

Description

The contract has a variable named `owner` that is initialized to zero address, creating ambiguity regarding the ownership structure. As a result, one might assume that the contract ownership is renounced. However, the contract uses an access control library that includes an admin role serving as the default owner.

This leads to confusion and misunderstanding about the actual owner of the contract.

```
address public owner = address(0);

_setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
```

Recommendation

It is recommended to remove the redundant `owner` variable and clearly document the usage of the access control library to define and manage ownership in the contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/Papi.sol#L104,105,111
Status	Unresolved

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.

```
name
symbol
distribution
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Papi.sol#L22
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.

```
address public owner = address(0)
```

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	contracts/PapiDistribution.sol#L14contracts/Papi.sol#L24,125,129,134,138,143,153,158,167,176,182,188,208,215,229,239,249,265,271,291,317,343@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#L5
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.

```
uint256 _amount
address _to
address _token
address public constant deadAddress =
0x000000000000000000000000000000000000dEaD
address _account
address _recipient
address _owner
address _spender
address _sender
uint256 _addedValue
uint256 _subtractedValue
address _lpToken
bool _lp
bool _isExcludedFromFee

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	@openzeppelin/contracts/utils/Strings.sol#L15,40@openzeppelin/contracts/access/AccessControl.sol#L206
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        //
https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c19824623
9e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
...
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/Papi.sol#L418,447
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.

```
uint256 _liquidityFeeHalf = liquidityFeeAmount / 2
uint256 _liquidityFeeToken1Amount = _calcFee(_token1Balance, _liquidityFeeHalf
* 10000 / _amountToSwap)
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/PapiDistribution.sol#L2contracts/Papi.sol#L2contracts/interfaces/IPapiDistribution.sol#L2@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol#L1@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#L1@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol#L1@openzeppelin/contracts/utils/Strings.sol#L4@openzeppelin/contracts/utils/introspection/IERC165.sol#L4@openzeppelin/contracts/utils/introspection/ERC165.sol#L4@openzeppelin/contracts/utils/Context.sol#L4@openzeppelin/contracts/token/ERC20/IERC20.sol#L4@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4@openzeppelin/contracts/access/IAccessControl.sol#L4@openzeppelin/contracts/access/AccessControl.sol#L4
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity >=0.5.0;
pragma solidity >=0.6.2;
pragma solidity ^0.8.2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/PapiDistribution.sol#L2contracts/Papi.sol#L2contracts/interfaces/IPapiDistribution.sol#L2@openzeppelin/contracts/utils/Strings.sol#L4@openzeppelin/contracts/utils/introspection/IERC165.sol#L4@openzeppelin/contracts/utils/introspection/ERC165.sol#L4@openzeppelin/contracts/utils/Context.sol#L4@openzeppelin/contracts/token/ERC20/IERC20.sol#L4@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4@openzeppelin/contracts/access/IAccessControl.sol#L4@openzeppelin/contracts/access/AccessControl.sol#L4
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 ^0.8.2;
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/PapiDistribution.sol#L15
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.

```
IERC20(_token).transfer(_to, _amount)
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AccessControl	Implementation	Context, IAccessControl, ERC165		
	supportsInterface	Public		-
	hasRole	Public		-
	_checkRole	Internal		
	_checkRole	Internal		
	getRoleAdmin	Public		-
	grantRole	Public	✓	onlyRole
	revokeRole	Public	✓	onlyRole
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
	_setRoleAdmin	Internal	✓	
	_grantRole	Internal	✓	
	_revokeRole	Internal	✓	
IAccessControl	Interface			
	hasRole	External		-
	getRoleAdmin	External		-
	grantRole	External	✓	-

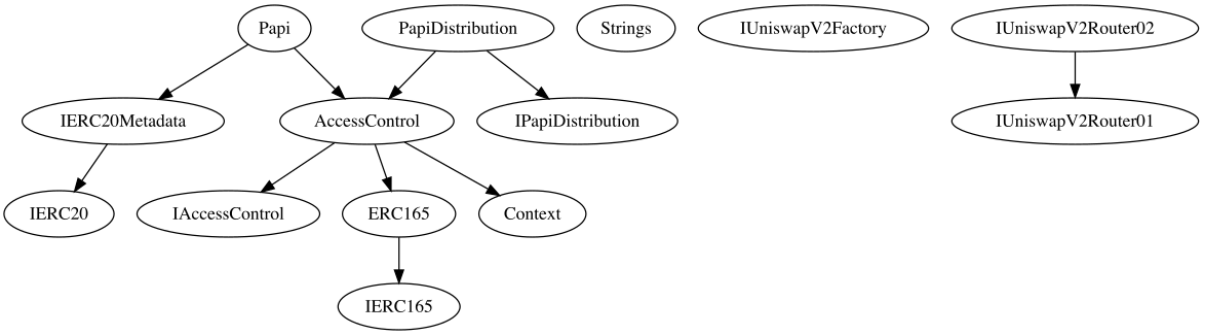
	revokeRole	External	✓	-
	renounceRole	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

IERC165	Interface			
	supportsInterface	External		-
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
Papi	Implementation	IERC20Metadata, AccessControl		
		Public	✓	-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	Public		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	setLpToken	External	✓	onlyRole
	setExcludedFromFee	Public	✓	onlyRole
	setExcludedFromSwap	Public	✓	onlyRole
	setRewardSwapReceivers	External	✓	onlyRole
	setRewardSellRate	External	✓	onlyRole

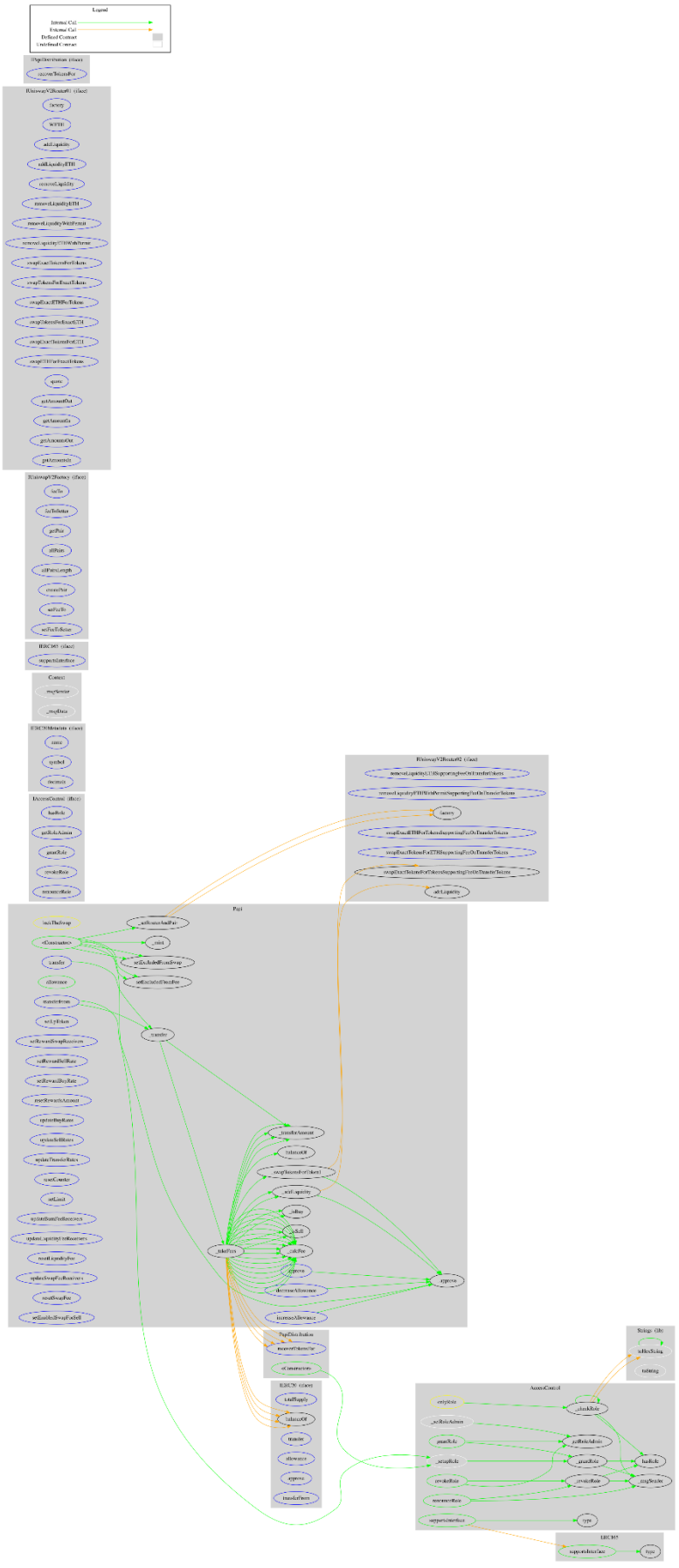
	setRewardBuyRate	External	✓	onlyRole
	resetRewardsAmount	External	✓	onlyRole
	updateBuyRates	External	✓	onlyRole
	updateSellRates	External	✓	onlyRole
	updateTransferRates	External	✓	onlyRole
	resetCounter	External	✓	onlyRole
	setLimit	External	✓	onlyRole
	updateBurnFeeReceivers	External	✓	onlyRole
	updateLiquidityFeeReceivers	External	✓	onlyRole
	resetLiquidityFee	External	✓	onlyRole
	updateSwapFeeReceivers	External	✓	onlyRole
	resetSwapFee	External	✓	onlyRole
	setEnabledSwapForSell	External	✓	onlyRole
	_transfer	Internal	✓	
	_takeFees	Internal	✓	
	_transferAmount	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
	_setRouterAndPair	Internal	✓	
	_calcFee	Internal		
	_isSell	Internal		
	_isBuy	Internal		
	_swapTokensForToken1	Internal	✓	lockTheSwap

	_addLiquidity	Internal	✓	lockTheSwap
PapiDistribution	Implementation	IPapiDistribution, AccessControl		
		Public	✓	-
	recoverTokensFor	External	✓	onlyRole

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router02
IUniswapV2Factory
IERC165
Context
IERC20Metadata
IAccessControl
Papi
PapiDistributor
IERC20
AccessControl
Strings
IERC165

Summary

Papi Coin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Papi Coin is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of a max 9% fee. Additionally, the contract has a fee limit mechanism. Moreover, it appears that the contract is falsely represented as being renounced, when in fact, it has not undergone the renouncement process.

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