



Cyberscope

# Audit Report

## **Banana Pepe**

May 2023

Network    BSC

Address    0xbe7eb1fde95912d19742d4d821545cea154b23d3

Audited by    © cyberscope

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Review</b>	<b>3</b>
Audit Updates	3
Source Files	3
<b>Findings Breakdown</b>	<b>4</b>
<b>Analysis</b>	<b>5</b>
<b>Diagnostics</b>	<b>6</b>
IDI - Immutable Declaration Improvement	7
Description	7
Recommendation	7
L04 - Conformance to Solidity Naming Conventions	8
Description	8
Recommendation	9
L07 - Missing Events Arithmetic	10
Description	10
Recommendation	10
L09 - Dead Code Elimination	11
Description	11
Recommendation	11
L12 - Using Variables before Declaration	12
Description	12
Recommendation	12
L13 - Divide before Multiply Operation	13
Description	13
Recommendation	13
L14 - Uninitialized Variables in Local Scope	14
Description	14
Recommendation	14
L16 - Validate Variable Setters	15
Description	15
Recommendation	15
L20 - Succeeded Transfer Check	16
Description	16
Recommendation	16
<b>Functions Analysis</b>	<b>17</b>
<b>Inheritance Graph</b>	<b>22</b>
<b>Flow Graph</b>	<b>23</b>
<b>Summary</b>	<b>24</b>
<b>Disclaimer</b>	<b>25</b>



## Review

Contract Name	BananaPepe
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	500 runs
Explorer	<a href="https://bscscan.com/address/0xbe7eb1fde95912d19742d4d821545cea154b23d3">https://bscscan.com/address/0xbe7eb1fde95912d19742d4d821545cea154b23d3</a>
Address	0xbe7eb1fde95912d19742d4d821545cea154b23d3
Network	BSC
Symbol	BAPEPE
Decimals	9
Total Supply	420,000,000,000,000

## Audit Updates

Initial Audit	18 May 2023
Corrected Phase 2	20 May 2023

## Source Files

Filename	SHA256
BananaPepe.sol	aa2aab44ece400961a655e73e196eebcf27d1832346a166f147fe333da83151b

## Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

# 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
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

## IDI - Immutable Declaration Improvement

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L233
<b>Status</b>	Unresolved

### Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
BUSD
```

### 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.



## L04 - Conformance to Solidity Naming Conventions

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L33,121,124,133,134,135,136,137,154,160,169,170,171,172,178,187,208,484,564
<b>Status</b>	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.

```
function WETH() external pure returns (address);
mapping (address => uint256) _tOwned
mapping (address => mapping (address => uint256)) _allowances
uint256 constant private startingSupply = 420_000_000_000_000
string constant private _name = "Banana Pepe"
string constant private _symbol = "BAPEPE"
uint8 constant private _decimals = 9
uint256 constant private _tTotal = startingSupply * (10 **
_decimals)

Fees public _taxRates = Fees({
    buyFee: 700,
    sellFee: 700,
    transferFee: 0
})

...
```

## 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L523,528,544,554,571
<b>Status</b>	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.

```
_maxTxAmount = (_tTotal * percent) / divisor  
_maxWalletSize = (_tTotal * percent) / divisor  
swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor  
piSwapPercent = priceImpactSwapPercent  
cashierGas = gas
```

### 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.

## L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BananaPepe.sol#L610
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 _basicTransfer(address from, address to, uint256 amount)
internal returns (bool) {
    _tOwned[from] -= amount;
    _tOwned[to] += amount;
    emit Transfer(from, to, amount);
    return true;
}
```

### 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.

## L12 - Using Variables before Declaration

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L774,786
<b>Status</b>	Unresolved

### Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 initThreshold  
uint256 initSwapAmount  
bool check
```

### Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

## L13 - Divide before Multiply Operation

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L680,698,845,846
<b>Status</b>	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 precision.

```
uint256 feeAmount = amount * currentFee / masterTaxDivisor
uint256 burnAmt = (feeAmount * ratios.burn) / total
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L774,785,786
<b>Status</b>	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 initThreshold  
uint256 initSwapAmount  
bool checked  
bool check
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L322
<b>Status</b>	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.

```
operator = newOperator
```

### 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.



## L20 - Succeeded Transfer Check

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BananaPepe.sol#L743,746,882
<b>Status</b>	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.

```
try _BUSD.transfer(_taxWallets.marketing, marketingBalance) {}  
catch {}  
try _BUSD.transfer(_taxWallets.rewardsPool, rewardsPoolBalance) {}  
catch {}  
TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)))
```

### 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
<b>Protections</b>	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	getInits	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-
	removeBlacklisted	External	✓	-
	isBlacklisted	External		-
	transfer	External	✓	-
	setBlacklistEnabled	External	✓	-
	setBlacklistEnabledMultiple	External	✓	-
<b>Cashier</b>	Interface			
	setRewardsProperties	External	✓	-
	tally	External	✓	-
	tallyZero	External	✓	-
	load	External	Payable	-

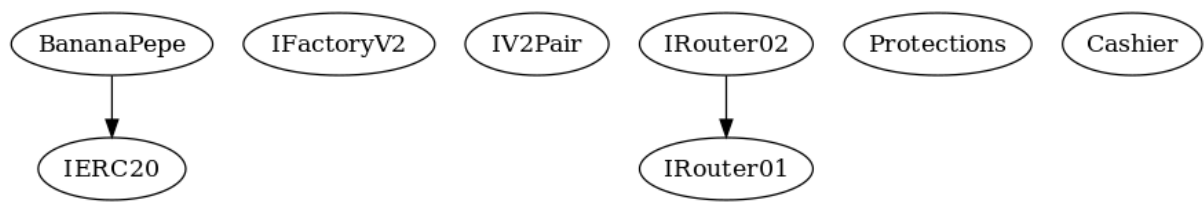
	cashout	External	✓	-
	giveMeWelfarePlease	External	✓	-
	getTotalDistributed	External		-
	getUserInfo	External		-
	getUserRealizedRewards	External		-
	getPendingRewards	External		-
	initialize	External	✓	-
	getCurrentReward	External		-
<b>BananaPepe</b>	Implementation	IERC20		
		Public	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-

	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	Public	✓	onlyOwner
	transfer	External	✓	-
	transferFrom	External	✓	-
	setNewRouter	External	✓	onlyOwner
	setLpPair	External	✓	onlyOwner
	setInitializers	External	✓	onlyOwner
	isExcludedFromFees	External		-
	isExcludedFromDividends	External		-
	isExcludedFromProtection	External		-
	isExcludedFromLimits	External		-
	setExcludedFromLimits	External	✓	onlyOwner
	setDividendExcluded	Public	✓	onlyOwner
	setExcludedFromFees	Public	✓	onlyOwner
	setExcludedFromProtection	External	✓	onlyOwner
	setBlacklistEnabled	External	✓	-
	setBlacklistEnabledMultiple	External	✓	-
	isBlacklisted	External		-
	removeBlacklisted	External	✓	onlyOwner
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner

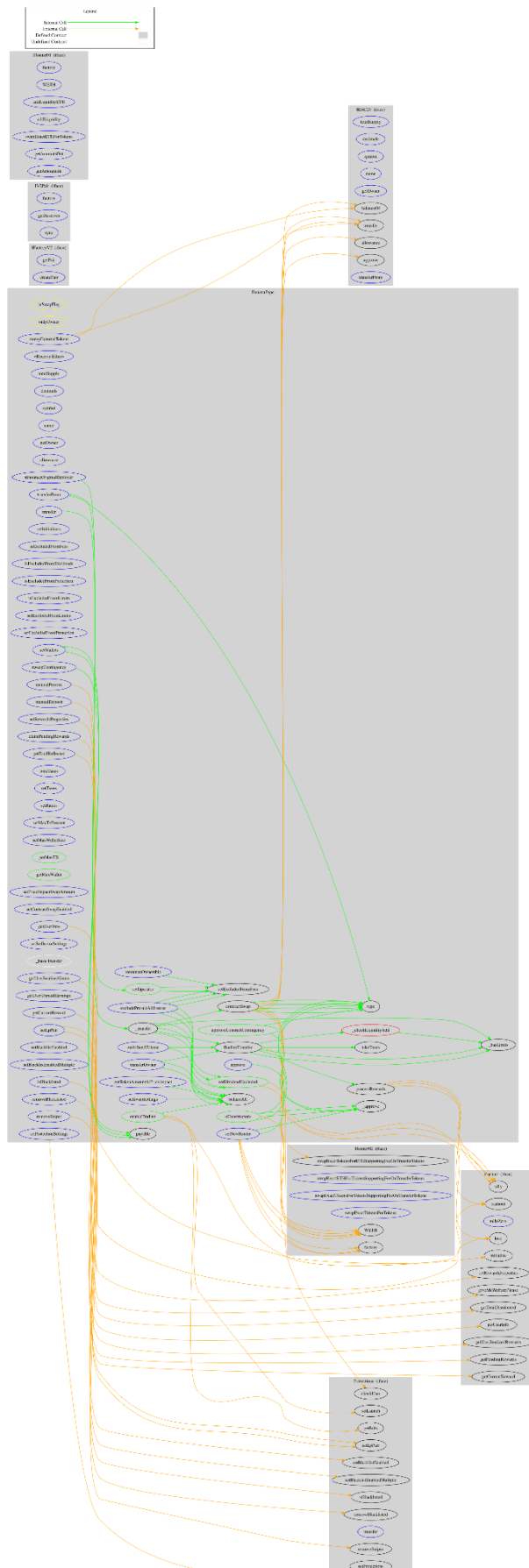
	lockTaxes	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setRatios	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	setMaxWalletSize	External	✓	onlyOwner
	getMaxTX	Public		-
	getMaxWallet	Public		-
	getTokenAmountAtPriceImpact	External		-
	setSwapSettings	External	✓	onlyOwner
	setPriceImpactSwapAmount	External	✓	onlyOwner
	setContractSwapEnabled	External	✓	onlyOwner
	setRewardsProperties	External	✓	onlyOwner
	setReflectorSettings	External	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_basicTransfer	Internal	✓	
	_transfer	Internal	✓	
	contractSwap	Internal	✓	inSwapFlag
	_checkLiquidityAdd	Private	✓	
	enableTrading	Public	✓	onlyOwner
	finalizeTransfer	Internal	✓	
	processRewards	Internal	✓	
	manualProcess	External	✓	-

	takeTaxes	Internal	✓	
	multiSendTokens	External	✓	onlyOwner
	manualDeposit	External	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	claimPendingRewards	External	✓	-
	getTotalReflected	External		-
	getUserInfo	External		-
	getUserRealizedGains	External		-
	getUserUnpaidEarnings	External		-
	getCurrentReward	External		-

## Inheritance Graph



# Flow Graph





## Summary

Banana Pepe contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. banana Pepe is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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 max 7% fee.

## 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>