Cyberscope

Audit Report

Bounty Square Ecosystem

February 2023

Type	BEP20
Network	BSC
Address	0x763cc90ba6a9fd473ce15624968fd56f79cc4447
Audited by	© cyberscope

Table of Contents

Review

Contract Name	BountySquareEcosystem
Compiler Version	v0.8.12+commit.f00d7308
Optimization	999 runs
Explorer	https://bscscan.com/address/0x763cc90ba6a9fd473ce15624968fd56f79cc4447
Address	0x763cc90ba6a9fd473ce15624968fd56f79cc4447
Network	BSC
Symbol	byne
Decimals	18
Total Supply	100,000,000

Audit Updates

Initial Audit	14 Feb 2023 https://github.com/cyberscope-io/audits/tree/main/bset/v1/audit.pdf
Corrected Phase 2	17 Feb 2023

Source Files

Filename	SHA256
BountySquareEcosystem.sol	4a70332a852a0ab553c0cd6a79a40e8d3940784624ee9381983f004714250b28

Introduction

Bounty Square Ecosystem implements a BEP20 token functionality enriched with a multisig pattern. This audit focuses on the BountySQEcosys.sol and the MultiSignAuth.sol files since all the other files are contracts from well-known trusted sources.

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
●	PVC	Price Volatility Concern	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	MMF	Misleading Multisig Functionality	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	BountySQEcosys.sol#L836
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapBackSettings(bool _enabled, uint256 _threshold, uint256
_pcThresholdMaxSell) external onlyOwners {
    require(pcThresholdMaxSell >= 100, "The _pcThresholdMaxSell has to be
100 or higher");

    swapEnabled = _enabled;
    swapThreshold = _threshold;
    pcThresholdMaxSell = _pcThresholdMaxSell;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	BountySQEcosys.sol#L771
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 amountBNBLiquidity = amountBNB.mul(liquidityFee).div(buySellFee);
uint256 amountBNBDev = amountBNB.mul(devFee).div(buySellFee);
uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(buySellFee);
uint256 amountBNBRewards = amountBNB.mul(rewardsFee).div(buySellFee);
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	BountySQEcosys.sol#L776
Status	Unresolved

Description

The contract sends funds to a `marketingFeeReceiver`, `devFeeReceiver`,`rewardsFeeReceiver` and `liquidityFeeReceiver` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingFeeReceiver).transfer(amountBNBMarketing);
payable(devFeeReceiver).transfer(amountBNBDev);
payable(rewardsFeeReceiver).transfer(amountBNBRewards);
payable(liquidityFeeReceiver).transfer(amountBNBLiquidity);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BountySQEcosys.sol#L12
Status	Unresolved

Description

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

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

```
using SafeMath for uint256;
```

Recommendation

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

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

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

MMF - Misleading Multisig Functionality

Criticality	Minor / Informative
Status	Unresolved

Description

The contract implements a multi-sig mechanism for the burn and trade enable functionalities. The minimum number of signers is one. Hence, if one signer is only registered, then it loses the multi-sig purpose.

Recommendation

The team is advised to add a restriction to the minimum number of acceptable signers to run a functional multi-sig wallet. The multi-sig mechanism could also implement a time-locker so the users will be able to track the changes in an acceptable timeframe.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BountySquareEcosystem.sol#L602,603,611,624,625,628,629,630,631
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 DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
uint256 public _totalSupply = 100_000_000 * (10 ** _decimals)
uint256 public buySellFee = 400
uint256 public feeDenominator = 10000
uint256 public liquidityFee = 175
uint256 public marketingFee = 100
uint256 public rewardsFee = 50
uint256 public devFee = 75
```

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	BountySquareEcosystem.sol#L246,394,602,603,604,605,607,608,609,611,612,614,615,812,832,848
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.

```
function WETH() external pure returns (address);
address[] memory _owners
uint _required

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	BountySquareEcosystem.sol#L645
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 public swapThreshold = _totalSupply / 1000 * 3
```

Recommendation

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

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	BountySquareEcosystem.sol#L500,515
Status	Unresolved

Description

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

```
uint count
```

Recommendation

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BountySquareEcosystem.sol#L6
Status	Unresolved

Description

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

```
pragma solidity ^0.8.9;
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	BountySquareEcosystem.sol#L852
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.

```
IBEP20(_token).transfer(msg.sender, _contractBalance)
```

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

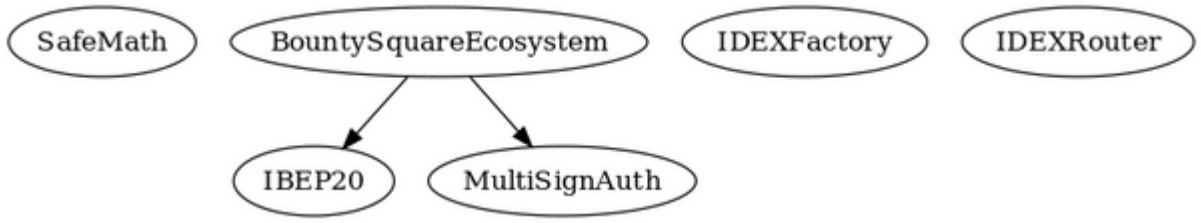
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-

	approve	External	✓	-
	transferFrom	External	✓	-
	burn	External	✓	-
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
MultiSignAuth	Implementation			
		Public	✓	-
	MultiSignOwners	Public	✓	multiSignReq validRequirement
	revokeConfirmation	Public	✓	ownerExists confirmed notExecuted
	addTransaction	Internal	✓	
	confirmTransaction	Internal	✓	ownerExists transactionExists notConfirmed
	submitTransaction	Internal	✓	
	multiSign	Internal	✓	
	getConfirmationCount	Public		-

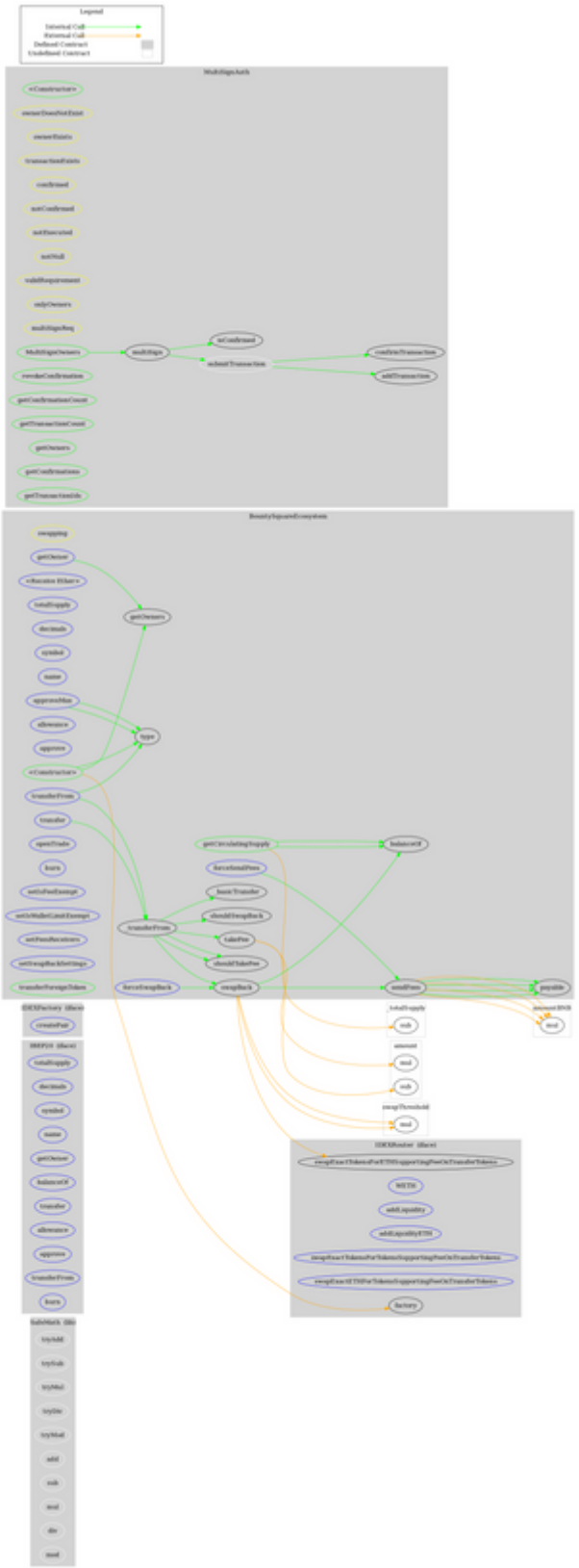
	getTransactionCount	Public		-
	isConfirmed	Public		-
	getOwners	Public		-
	getConfirmations	Public		-
	getTransactionIds	Public		-
BountySquare Ecosystem	Implementation	IBEP20, MultiSignAuth		
		Public	✓	MultiSignAuth
		External	Payable	-
	getCirculatingSupply	Public		-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	External	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	sendFees	Internal	✓	
	openTrade	External	✓	onlyOwners

	burn	External	✓	-
	setIsFeeExempt	External	✓	onlyOwners
	setIsWalletLimitExempt	External	✓	onlyOwners
	setFeesReceivers	External	✓	onlyOwners
	setSwapBackSettings	External	✓	onlyOwners
	forceSwapBack	External	✓	onlyOwners
	forceSendFees	External	✓	onlyOwners
	transferForeignToken	Public	✓	onlyOwners

Inheritance Graph



Flow Graph



Summary

Bounty Square Ecosystem 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.

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