BSCHEX

WEBSITE

https://www.bschex.xyz

TELEGRAM

https://t.me/BSCHexchat

SMART CONTRACT AUDIT

Twitter:
twitter.com/vital_blockA

Telegram:
t.me/vital_Block

Disclaimer

Vital Block Solidity reports are not, nor should be considered, an "endorsement" or "disapproval" of any project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Vital Block to perform a security review.

Vital Black Solidity Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analysed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

Vital Block Solidity Reports should not be used in any way to make decisions around investment or involvement with any project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. Vital Block Solidity Reports represent an extensive auditing 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. Vital Block Solidity's position is that each company and individual are responsible for their own due diligence and continuous security. Vital Block Solidity'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 analyse

What is a Vital Block Audit report?

•A document describing in detail an in-depth analysis of a particular piece(s) of source code provided to Vital Block Solidity by a Client.

•An organized collection of testing results, analysis and inferences made about the structure, implementation, and overall best practices of a particular piece of source code.

•Representation that a Client of Vital Block Solidity has indeed completed a round of auditing with the intention to increase the quality of the company/ product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	**BSCHEX TOKEN**
Description	**BSChex is blockchain's first entertainment aggregator and licensed producer of original motion picture, exclusive Pay-Per-View events, and surreal gaming experiences.**
Platform	**Binance Mainnet**
Mainnet Contracts:	**0xdDD66D900A9f42f32774D9Cb28C5E9C7c2aCDA91** ***BSCHEX TOKEN* (BSCHEX)**
Files:	BSCHEX.sol

Audit Summary

Delivery Date	**August 23 2022**
Method of Audit	**Security Static Analysis**
Timeline	**Story Points 100**

Vulnerability Summary

Total Issues Found	**0**
Total Issues Resolved	**0**
Total Critical	**0**
Total High	**1**
Total Medium	**2**
Total Low	**0**
Total Informational	**2**

Our Audit Methodology

- **STEP 1**
  A manual line-by-line code review to ensure the logic behind each function is safe and secured against common attack vectors.

- **STEP 2**
  Simulation of hundreds of thousands of Smart Contract Interactions on a test and Mainnet blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.

- **STEP 3**
  Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.

Grading

The following grading structure is used to assess the level of vulnerability found within all Smart Contracts:

THREAT LEVEL	DEFINITION
Critical	Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss.
High	Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Medium	Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Low	Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract.
Informational	Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts

Description

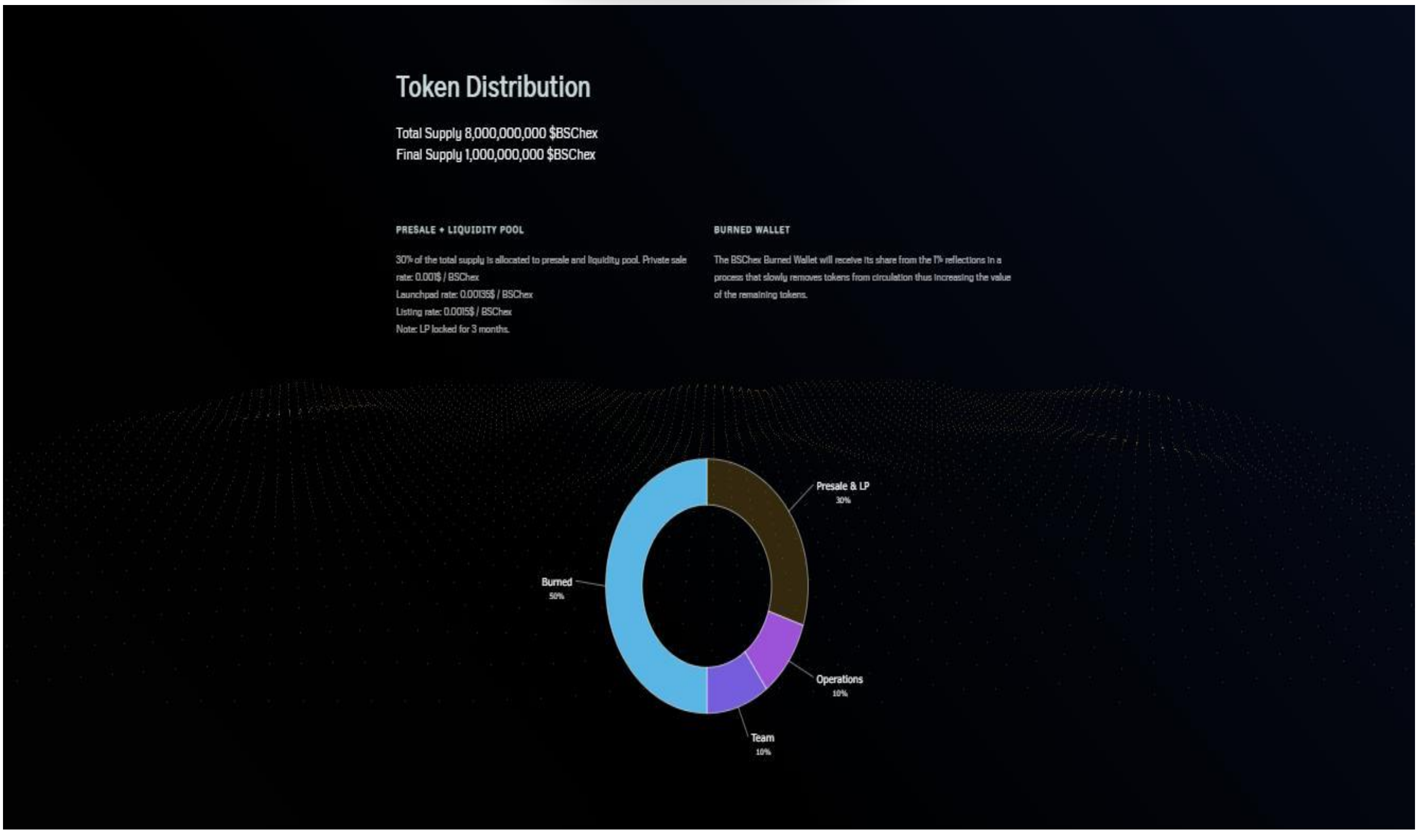
BSCHEX : The (BSChex) Protocol charges 12% transaction fees and it is distributed to 3 main features: token reflections to all holders, buy-back (we call it the Robin Hood System) and crypto collateral asset rewards to be claimed by holders from our rewards Dapp.

Buy Trading Fees 12. 2%-LP | 4% Buy Back | 4% BNB Reflection | 2% BSChex Distribution

Sell Trading Fees 12. 2%-LP | 4% Buy Back | 4% BNB Reflection | 2% BSChex Distribution

Initial supply : **1,000,000,000 BSCHEX**

BSCHEX 🌎 DISTRIBUTION



Vulnerability 0: No important security issue detected.
Threat level: Low

Description:
Not a honeypot transaction simulation is success at the moment. Always DYOR before investing.

INFO! There is no liquidity with BNB. Results with non-BNB pair may differ. If the token is not live yet, results may be different once the token is live. It is common for tokens to have 0% taxes before launching on DEX!

Home | BSCHEX.sol

```
// Base class that implements: BEP20 interface, fees & swaps
abstract contract aBase is Context, IERC20Metadata, Ownable, ReentrancyGuard {
    // MAIN TOKEN PROPERTIES
    string private constant NAME = "BSCHex - P2E Metaverse";
    string private constant SYMBOL = "BSCHEX";
    uint8 private constant DECIMALS = 9;
    uint8 private _liquidityFee; //% of each transaction that will be added as liquidity
    uint8 private _rewardFee; //% of each transaction that will be used for BNB reward pool
    uint8 private _additionalSellFee; //Additional % fee to apply on sell transactions. Half of it will go to liquidity, other half to rewards
    uint8 private _poolFee; //The total fee to be taken and added to the pool, this includes both the liquidity fee and the reward fee

    uint256 private constant _totalTokens = 1000000000000 * 10**DECIMALS;   //1 trillion total supply
    mapping (address => uint256) private _balances; //The balance of each address.  This is before applying distribution rate.  To get the actu
    mapping (address => mapping (address => uint256)) private _allowances;

    // FEES & REWARDS
    bool private _isSwapEnabled; // True if the contract should swap for liquidity & reward pool, false otherwise
    bool private _isFeeEnabled; // True if fees should be applied on transactions, false otherwise
    address public constant BURN_WALLET = 0x000000000000000000000000000000000000dEaD; //The address that keeps track of all tokens burned
    uint256 private _tokenSwapThreshold = _totalTokens / 100000; //There should be at least 0.0001% of the total supply in the contract before
    uint256 private _totalFeesPooled; // The total fees pooled (in number of tokens)
    uint256 private _totalBNBLiquidityAddedFromFees; // The total number of BNB added to the pool through fees
    mapping (address => bool) private _addressesExcludedFromFees; // The list of addresses that do not pay a fee for transactions

    // TRANSACTION LIMIT
    uint256 private _transactionLimit = _totalTokens; // The amount of tokens that can be sold at once
    bool private _isBuyingAllowed; // This is used to make sure that the contract is activated before anyone makes a purchase on PCS.  The cont
```

0 listen on all transactions | Search with transaction hash or address

Vulnerability 1: The owner of this smart-contract can modify the trading fees of the token

Threat level: Low

Vulnerability 1: Gas optimisation

Threat level 2: Informational

Description: this smart-contract can be Modified by Deployer
This can always change! Do your own due diligence.
INFO! Owner can change trading tax fee. which is Really a normal function for most Smart Contract. Removal fee is private and calculate function

BSCHEX TOKEN (BSCHEX)

No trading data available: either trading is disabled, or no Liquidity for the token Yet.

Recommendation:
The ownership of the contract isn't renounced..

```
//to recieve ETH from uniswapV2Router when swaping
receive() external payable {}

function _reflectFee(uint256 rFee, uint256 tFee) private {
    _rTotal = _rTotal.sub(rFee);
    _tFeeTotal = _tFeeTotal.add(tFee);
}

function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, u
    (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getTValues(tAmount);
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFe
    return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity);
}

function _getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256)
    uint256 tFee = calculateTaxFee(tAmount);
    uint256 tLiquidity = calculateLiquidityFee(tAmount);
    uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
    return (tTransferAmount, tFee, tLiquidity);
}

function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 current
    uint256 rAmount = tAmount.mul(currentRate);
    uint256 rFee = tFee.mul(currentRate);
    uint256 rLiquidity = tLiquidity.mul(currentRate);
    uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
    return (rAmount, rTransferAmount, rFee);
}

function _getRate() private view returns(uint256) {
    (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
    return rSupply.div(tSupply);
```

Issues Checking Status

	Issue description	Checking status
1.	Compiler errors.	Passed
2.	Race conditions and Reentrancy. Cross-function race conditions.	Passed
3.	Possible delays in data delivery.	Passed
4.	Oracle calls.	Passed
5.	Front running.	Passed
6.	Timestamp dependence.	Passed
7.	Integer Overflow and Underflow.	Passed
8.	DoS with Revert.	Passed
9.	DoS with block gas limit.	Passed
10.	Methods execution permissions.	Passed
11.	Economy model of the contract.	Passed
12.	The impact of the exchange rate on the logic.	Passed
13.	Private user data leaks.	Passed
14.	Malicious Event log.	Passed
15.	Scoping and Declarations.	Passed
16.	Uninitialized storage pointers.	Passed
17.	Arithmetic accuracy.	Passed
18.	Design Logic.	Passed
19.	Cross-function race conditions.	Passed
20.	Safe Open Zeppelin contracts implementation and usage.	Passed
21.	Fallback function security.	Passed

BSCHEX

Audit Result

PASSED

Conclusion

During the Vital block Audit process, the BSCHEX contract was analysed by manual review and automated testing. All issues identified was after deployment to mainnet. By submitting the contract for audit after Deployment, the team have displayed a strong commitment to security.

Whilst there are no obvious vulnerabilities or security risks identified within the main net contract, it is beyond the scope of this Vital Block Audit to comment upon any risks associated with tokenomics, adoption or platform longevity. Before placing funds in any defi protocol Vital Block encourages potential investors to exercise due diligence and research all projects thoroughly to assess plans for ongoing development and financial sustainability.

Appendix

Finding Categories

Gas Optimization
Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations
Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue
Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow
Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code
Volatile Code findings refer to segments of code that behave unexpectely on certain edge cases that may result in avulnerability.

Data Flow
Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment operation affecting an in-memory struct rather than an instorage one.

Language Specific
Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style
Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Appendix

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

VitalBlock

Making Defi And Web3 a Safer place

WWW.VITALBLOCK.ORG

Decentralized Smart Contract Auditing Firm.