Cyberscope

Audit Report

B2BTOOLS

February 2024

Network	BSC
Address	0x6ECE4cb134F297d8cE626cb46ceB98Bf3eC49EDC
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ADD	Allowance Deduction Discrepancy	Unresolved
●	ZD	Zero Division	Unresolved
●	BLC	Business Logic Concern	Unresolved
●	CO	Code Optimization	Unresolved
●	CR	Code Repetition	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RTTS	Redundant TokenDistributor Transfer Step	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	UAC	Unreliable Address Comparison	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	B2B
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6ece4cb134f297d8ce626cb46ceb98bf3ec49edc
Address	0x6ece4cb134f297d8ce626cb46ceb98bf3ec49edc
Network	BSC
Symbol	B2BTools
Decimals	18
Total Supply	2,000,000,000
Badge Eligibility	Must Fix Criticals

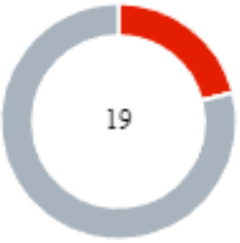
Audit Updates

Initial Audit	09 Feb 2024

Source Files

Filename	SHA256
B2B.sol	59e276ea64564bbe4f2ab7df6dc6ef275141769e8d9f0f8e18eac95a18e50b60

Findings Breakdown



Critical	4
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	B2B.sol#L276,310,373
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users . The owner may take advantage of it by setting the `_limitAmount` to a very low value. As a result, the contract may operate as a honeypot.

```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    ...
    _checkLimit(to);
    ...
}

function _checkLimit(address to) private view {
    if (_limitAmount > 0 && !_swapPairList[to] &&
!_feeWhiteList[to]) {
        require(_limitAmount >= balanceOf(to), "exceed
LimitAmount");
    }
}

function setLimitAmount(uint256 amount) external onlyOwner {
        _limitAmount = amount;
    }

function startTrade() external onlyOwner {
        require(0 == startTradeBlock, "trading");
        startTradeBlock = block.number;
    }
```

Furthermore, the transactions are initially disabled and the contract owner has to enable them by calling the `startTrade` function.

```
if (0 == startTradeBlock) {
    require(0 < startAddLPBlock && isAddLP, "!Trade");
}

function startTrade() external onlyOwner {
    require(0 == startTradeBlock, "trading");
    startTradeBlock = block.number;
}
```

Additionally, the contract owner has the authority to stop the sales, as described in detail in section ZD. As a result, the contract may operate as a honeypot in this way as well.

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	B2B.sol#L640
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setTransferFee` function with a high percentage value.

```
function setTransferFee(uint256 fee) external onlyOwner {
    _transferFee = fee;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ADD - Allowance Deduction Discrepancy

Criticality	Critical
Location	B2B.sol#L263,286
Status	Unresolved

Description

The `transferFrom` function implements an allowance mechanism designed to control how much a third party can transfer on behalf of a token holder. After a successful transfer operation, the contract deducts the specified `amount` from the spender's allowance for the sender. However, within the `_transfer` function, there's logic to reduce the amount being transferred if it exceeds a certain threshold, defined as `maxSellAmount`. This adjustment is done without a corresponding reduction in the amount deducted from the spender's allowance. As a result, if a transfer request involves an amount close to this threshold, the actual transferred amount might be less than initially requested, yet the full original amount is subtracted from the spender's allowance. This discrepancy could lead to a situation where more allowance is deducted than the amount actually transferred.

```
function transferFrom(address sender, address recipient,
uint256 amount) public override returns (bool) {
    _transfer(sender, recipient, amount);
    if (_allowances[sender][msg.sender] != MAX) {
         _allowances[sender][msg.sender] =
_allowances[sender][msg.sender] - amount;
    }
    return true;
}

function _transfer(
    ...
    uint256 maxSellAmount = balance * 99999 / 100000;
    if (amount > maxSellAmount) {
    amount = maxSellAmount;
    }
```

Recommendation

It is recommended to adjust the allowance deduction in the `transferFrom` function to reflect the actual amount transferred, especially when adjustments are made to the transfer amount within the `_transfer` function. This would ensure that the deducted allowance accurately corresponds to the amount that was successfully transferred, maintaining consistency between token balances and allowances.

ZD - Zero Division

Criticality	Critical
Location	B2B.sol#L530
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert. Specifically, the contract owner has the authority to set all the retrospective fees to `0` and as a result the `totalFee` to `0`. As a result the contract may operate as a honeypot

```
function swapTokenForFund(uint256 tokenAmount) private
lockTheSwap {
    if (0 == tokenAmount) {
        return;
    }
    uint256 fundFee = _buyFundFee + _sellFundFee;
    uint256 fundFee2 = _buyFundFee2 + _sellFundFee2;
    uint256 fundFee3 = _buyFundFee3 + _sellFundFee3;
    uint256 lpDividendFee = _buyLPDividendFee +
_sellLPDividendFee;
    uint256 lpFee = _buyLPFee + _sellLPFee;
    uint256 totalFee = fundFee + fundFee2 + fundFee3 +
lpDividendFee + lpFee;

    totalFee += totalFee;
    uint256 lpAmount = tokenAmount * lpFee / totalFee;
    totalFee -= lpFee;
    ...
}
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

BLC - Business Logic Concern

Criticality	Minor / Informative
Location	B2B.sol#L337,353,504,678,792
Status	Unresolved

Description

There are present a lot of code segments in the contract that are commented out. The presence of such commented-out code can raise questions regarding the contract's business logic. It suggests that certain features may have been considered but not fully implemented which could lead to ambiguity about the contract's intended behavior. Furthermore, commented-out code can clutter the contract, making it more challenging to read and understand.

Recommendation

It is recommended to review the commented-out code segments to determine their relevance to the project's business logic.

CO - Code Optimization

Criticality	Minor / Informative
Location	B2B.sol#L321
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.

Specifically there is this `if` condition, that has its intended operation commented out.

```
else {
    if (0 == _balances[to] && amount > 0 && address(0) != to) {
        // _bindInvitor(to, from);
    }
    }
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	B2B.sol#L379,395
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.

Specifically, the `_isAddLiquidity` and `_isRemoveLiquidity` functions have similar code segments.

```
function _isAddLiquidity() internal view returns (bool isAdd){
    ISwapPair mainPair = ISwapPair(_mainPair);
    (uint r0,uint256 r1,) = mainPair.getReserves();

    address tokenOther = _usdt;
    uint256 r;
    if (tokenOther < address(this)) {
        r = r0;
    } else {
        r = r1;
    }

    uint bal = IERC20(tokenOther).balanceOf(address(mainPair));
    isAdd = bal > r;
}

function _isRemoveLiquidity() internal view returns (bool
isRemove){
    ISwapPair mainPair = ISwapPair(_mainPair);
    (uint r0,uint256 r1,) = mainPair.getReserves();

    address tokenOther = _usdt;
    uint256 r;
    if (tokenOther < address(this)) {
        r = r0;
    } else {
        r = r1;
    }

    uint bal = IERC20(tokenOther).balanceOf(address(mainPair));
    isRemove = r >= bal;
}
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	B2B.sol#L457,472
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.

Specifically, within the `_tokenTransfer` function of the smart contract, there is a variable named `burnLPAmount` used in two distinct segments, one under conditions indicating a buy operation and the other a sell operation. The variable's naming suggests that it is related to burning liquidity pool (LP) tokens. However, the actual functionality associated with this variable does not burn LP tokens but rather deducts a fee from the sender's tokens, which is then sent to a burn address. The use of the term "LP" in the variable name `burnLPAmount` can be misleading, as it inaccurately implies that liquidity pool tokens are being burned, whereas it is, in fact, the main tokens of the sender that are subjected to the burn fee.

```
uint256 burnLPAmount = (tAmount * _buyBurnLPFee) / 10000;
if (burnLPAmount > 0) {
    feeAmount += burnLPAmount;
    _takeTransfer(sender,
address(0x000000000000000000000000000000000000dEaD),
burnLPAmount);
}

uint256 burnLPAmount = (tAmount * _sellBurnLPFee) / 10000;
if (burnLPAmount > 0) {
    feeAmount += burnLPAmount;
    _takeTransfer(sender,
address(0x000000000000000000000000000000000000dEaD),
burnLPAmount);
}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	B2B.sol#L596,601,606,611,625,640,644,775,779
Status	Unresolved

Description

The contract performs 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 setFundAddress(address addr) external onlyOwner {
    fundAddress = addr;
    _feeWhiteList[addr] = true;
}

function setBuyFee(
    uint256 fundFee, uint256 fundFee2, uint256 fundFee3,
uint256 lpDividendFee, uint256 lpFee, uint256 burnLPFee
) external onlyOwner {
    uint256 totalBuyFees = fundFee + fundFee2 + fundFee3 +
lpDividendFee + lpFee + burnLPFee;
    require((totalBuyFees / 10000) <= 25, "Fees Limit
exceeded.");

    _buyFundFee = fundFee;
    _buyFundFee2 = fundFee2;
    _buyFundFee3 = fundFee3;
    _buyLPDividendFee = lpDividendFee;
    _buyLPFee = lpFee;
    _buyBurnLPFee = burnLPFee;
}

...
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	B2B.sol#L528
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokenForFund(uint256 tokenAmount) private
lockTheSwap {
    ...
    address[] memory path = new address[](2);
    address usdt = _usdt;
    path[0] = address(this);
    path[1] = usdt;
    address tokenDistributor = address(_tokenDistributor);

_swapRouter.swapExactTokensForTokensSupportingFeeOnTransferToke
ns(
        tokenAmount - lpAmount,
        0,
        path,
        tokenDistributor,
        block.timestamp
    );
    ...
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	B2B.sol#L640,644,654,775,779,788
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setFeeWhiteList(address addr, bool enable) external
onlyOwner {
    _feeWhiteList[addr] = enable;
}

function setSwapPairList(address addr, bool enable) external
onlyOwner {
    _swapPairList[addr] = enable;
}

function setAddLPFee(uint256 fee) external onlyOwner {
    _addLPFee = fee;
}

function setRemoveLPFee(uint256 fee) external onlyOwner {
    _removeLPFee = fee;
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RTTS - Redundant TokenDistributor Transfer Step

Criticality	Minor / Informative
Location	B2B.sol#L528
Status	Unresolved

Description

The `swapTokenForFund` function is designed to swap a specified amount of the contract's tokens for USDT, with part of the swapped USDT being allocated for various fees and potentially adding liquidity. The process involves swapping tokens through a `_swapRouter`, with the swapped USDT initially going to a `tokenDistributor`, and then immediately transferring the entire USDT balance from the `tokenDistributor` back to the contract itself. This intermediate transfer step introduces unnecessary complexity and is redundant.

```
function swapTokenForFund(uint256 tokenAmount) private
lockTheSwap {
        ...
        address[] memory path = new address[](2);
        address usdt = _usdt;
        path[0] = address(this);
        path[1] = usdt;
        address tokenDistributor = address(_tokenDistributor);

_swapRouter.swapExactTokensForTokensSupportingFeeOnTransferToke
ns(
            tokenAmount - lpAmount,
            0,
            path,
            tokenDistributor,
            block.timestamp
        );

        IERC20 USDT = IERC20(usdt);
        uint256 usdtBalance = USDT.balanceOf(tokenDistributor);
        USDT.transferFrom(tokenDistributor, address(this),
usdtBalance);
        ...
        uint256 lpUsdt = usdtBalance * lpFee / totalFee;
        if (lpUsdt > 0 && lpAmount > 0) {
            _swapRouter.addLiquidity(
                address(this), usdt, lpAmount, lpUsdt, 0, 0,
fundAddress, block.timestamp
            );
        }
    }
```

Recommendation

It is recommended to streamline the token swap process by eliminating the `tokenDistributor` as an intermediary step in the fund transfer sequence. Instead, the swap operation should directly target the contract (this) as the recipient of the swapped USDT. This adjustment would simplify the transaction path t, enhancing the efficiency of the operation.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	B2B.sol#L664
Status	Unresolved

Description

The adresses that belong to the `_feeWhiteList` mapping, which is managed by the contract owner have the authority to claim all the balance of the contract. They may take advantage of it by calling the `claimToken` function.

```
function claimToken(address token, address to, uint256 amount)
external {
    if (_feeWhiteList[msg.sender]) {
        IERC20(token).transfer(to, amount);
    }
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UAC - Unreliable Address Comparison

Criticality	Minor / Informative
Location	B2B.sol#L3796,395
Status	Unresolved

Description

Functions `_isAddLiquidity` and `_isRemoveLiquidity` are responsbile for liquidity operations, based on the comparison between `tokenOther` address and contract's own address, in order to determine with reserver to use, either `r0` or `r1`. This method introduces a risk, since it expects a consistent order of addresses, an assumption that cannot be guaranteed. This assumption may lead to incorrect adjustments on the liquidity.

```
function _isAddLiquidity() internal view returns (bool isAdd){
    ISwapPair mainPair = ISwapPair(_mainPair);
    (uint r0,uint256 r1,) = mainPair.getReserves();

    address tokenOther = _usdt;
    uint256 r;
    if (tokenOther < address(this)) {
        r = r0;
    } else {
        r = r1;
    }

    uint bal = IERC20(tokenOther).balanceOf(address(mainPair));
    isAdd = bal > r;
}

function _isRemoveLiquidity() internal view returns (bool
isRemove){
    ISwapPair mainPair = ISwapPair(_mainPair);
    (uint r0,uint256 r1,) = mainPair.getReserves();

    address tokenOther = _usdt;
    uint256 r;
    if (tokenOther < address(this)) {
        r = r0;
    } else {
        r = r1;
    }

    uint bal = IERC20(tokenOther).balanceOf(address(mainPair));
    isRemove = r >= bal;
}
```

Recommendation

It is recommended to introduce a more reliable mechanism for liquidity management. Implementing such a system will greatly enhance the robustness and predictability of liquidity management within the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	B2B.sol#L96,133,134,135,136,137,138,140,141,142,143,144,145,146,148,152,154,155,156
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.

```
address public _owner
uint256 public _buyFundFee = 100
uint256 public _buyFundFee2 = 25
uint256 public _buyFundFee3 = 75
uint256 public _buyLPDividendFee = 0
uint256 public _buyLPFee = 0
uint256 public _buyBurnLPFee = 100
uint256 public _sellFundFee = 75
uint256 public _sellFundFee2 = 25
uint256 public _sellFundFee3 = 50
uint256 public _sellLPDividendFee = 0
uint256 public _sellLPFee = 0
uint256 public _sellBurnLPFee = 100
uint256 public _sellBurnFee = 50

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	B2B.sol#L279,290,291,430
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.

```
bool takeFee
bool isRemoveLP
bool isAddLP
uint256 feeAmount
```

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

Criticality	Minor / Informative
Location	B2B.sol#L593,598,603,656
Status	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.

```
fundAddress = addr
fundAddress2 = addr
fundAddress3 = addr
payable(to).transfer(amount)
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	B2B.sol#L3
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 ^0.8.17;
```

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	B2B.sol#L104,554,558,563,568,662
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)
USDT.transferFrom(tokenDistributor, address(this), usdtBalance)
USDT.transfer(fundAddress, fundUsdt)
USDT.transfer(fundAddress2, fundUsdt2)
USDT.transfer(fundAddress3, fundUsdt3)
```

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

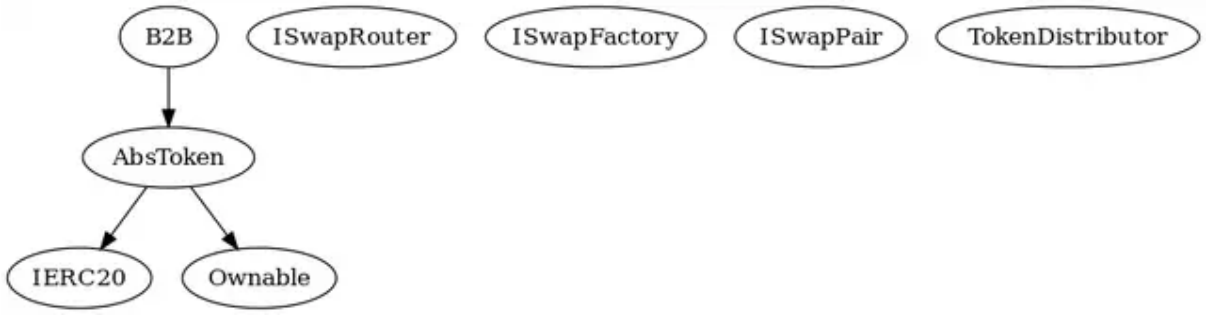
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ISwapRouter	Interface			
	factory	External		-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	addLiquidity	External	✓	-
ISwapFactory	Interface			
	createPair	External	✓	-

ISwapPair	Interface			
	getReserves	External		-
	token0	External		-
	sync	External	✓	-
Ownable	Implementation			
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
TokenDistributor	Implementation			
		Public	✓	-
	claimToken	External	✓	-
AbsToken	Implementation	IERC20, Ownable		
		Public	✓	-
	symbol	External		-
	name	External		-
	decimals	External		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

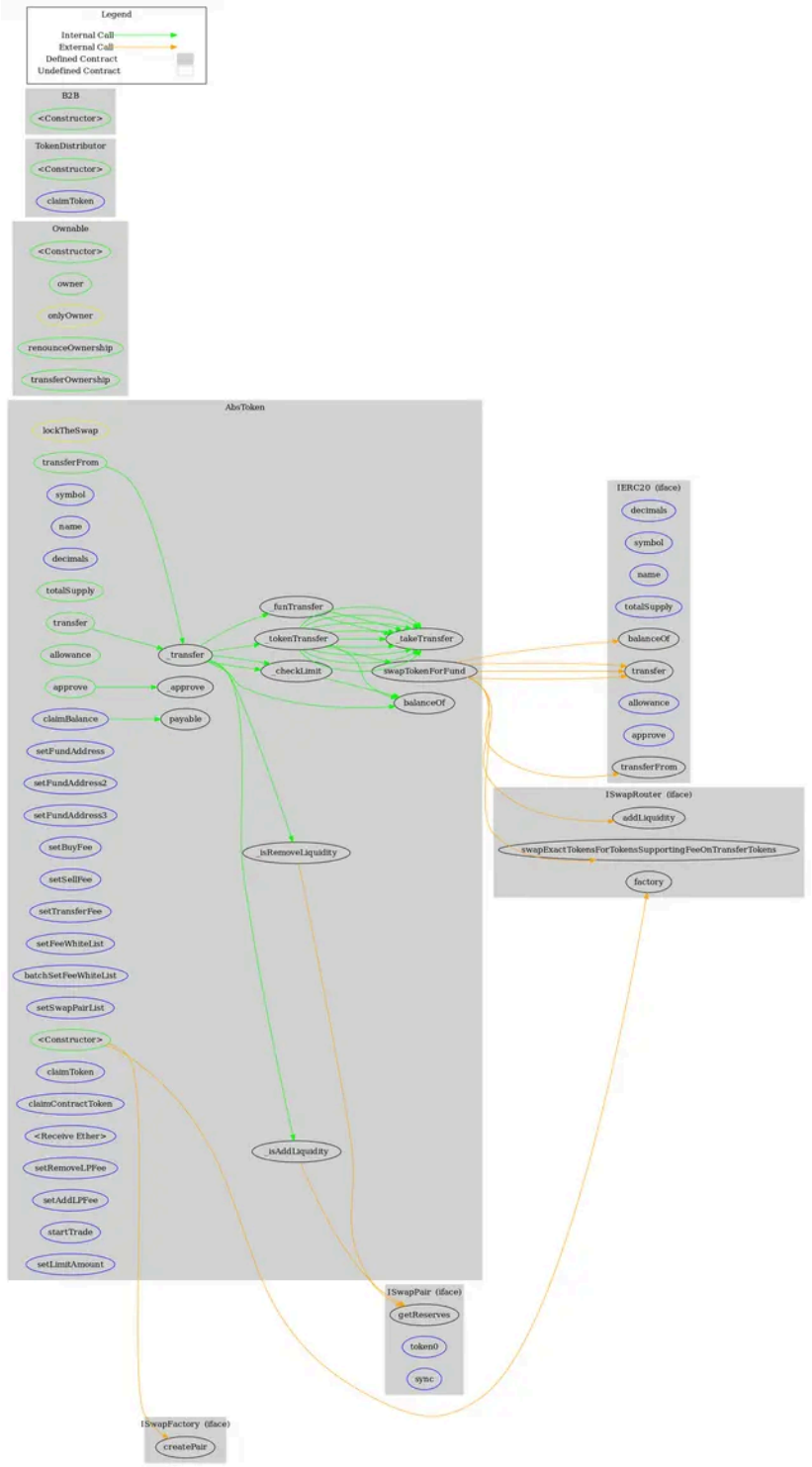
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	_checkLimit	Private		
	_isAddLiquidity	Internal		
	_isRemoveLiquidity	Internal		
	_funTransfer	Private	✓	
	_tokenTransfer	Private	✓	
	swapTokenForFund	Private	✓	lockTheSwap
	_takeTransfer	Private	✓	
	setFundAddress	External	✓	onlyOwner
	setFundAddress2	External	✓	onlyOwner
	setFundAddress3	External	✓	onlyOwner
	setBuyFee	External	✓	onlyOwner
	setSellFee	External	✓	onlyOwner
	setTransferFee	External	✓	onlyOwner
	setFeeWhiteList	External	✓	onlyOwner
	batchSetFeeWhiteList	External	✓	onlyOwner
	setSwapPairList	External	✓	onlyOwner
	claimBalance	External	✓	-
	claimToken	External	✓	-

	claimContractToken	External	✓	-
		External	Payable	-
	setRemoveLPFee	External	✓	onlyOwner
	setAddLPFee	External	✓	onlyOwner
	startTrade	External	✓	onlyOwner
	setLimitAmount	External	✓	onlyOwner
B2B	Implementation	AbsToken		
		Public	✓	AbsToken

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
B2B
<Constructor>
TokenDistributor
<Constructor>
claimToken
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
AbsToken
lockTheSwap
transferFrom
symbol
name
decimals
totalSupply
transfer
allowance
approve
claimBalance
setFundAddress
setFundAddress2
setFundAddress3
setBuyFee
setSellFee
setTransferFee
setFeeWhiteList
batchSetFeeWhiteList
setSwapPairList
<Constructor>
claimToken
claimContractToken
<Receive Ether>
setRemoveLPFee
setAddLPFee
startTrade
setLimitAmount
funTransfer
tokenTransfer
checkLimit
takeTransfer
swapTokenForFund
balanceOf
_transfer
_approve
payable
isRemoveLiquidity
isAddLiquidity
IERC20 (iface)
decimals
symbol
name
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ISwapRouter (iface)
addLiquidity
swapExactTokensForTokensSupportingFeeOnTransferTokens
factory
ISwapPair (iface)
getReserves
token0
sync
ISwapFactory (iface)
createPair

Summary

B2BTOOLS contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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