Cyberscope

Audit Report

BLOCKSPOT

November 2023

Network	ETH
Address	0x53020F42f6Da51B50cf6E23e45266ef223122376
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🟠	CCR	Contract Centralization Risk	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	MU	Modifiers Usage	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	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	BlockSpot
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://etherscan.io/address/0x53020f42f6da51b50cf6e23e45266ef223122376
Address	0x53020f42f6da51b50cf6e23e45266ef223122376
Network	ETH
Symbol	SPOT
Decimals	18
Total Supply	100,000,000

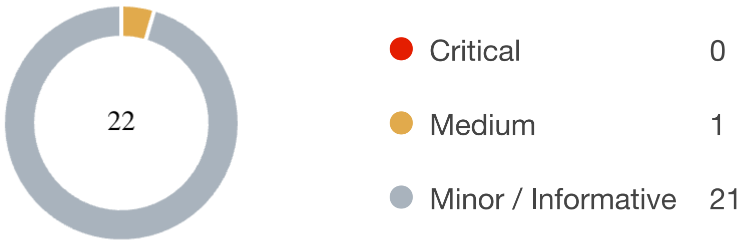
Audit Updates

Initial Audit	11 Nov 2023

Source Files

Filename	SHA256
contracts/SafeMath.sol	f39d9ee58c3ad0f46c8a1886875a5de9e833d8a97f6957075e18cf7ca6a3de27
contracts/LPDiv.sol	0b5f4d2826d379bc13ca93851a2e541450311365ca809f062ae27de932e0a7ab
contracts/ILPDiv.sol	4dc41822aeb1854ab7a9960879ae8a407c396435dcd4e56f88542f82318dbf3a
contracts/BlockSpot.sol	700125a605dfa2f9f83a0dfe1b37e075b9de5c8a748bb0725deee620de74e964
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/interfaces/IERC20.sol	1e78c90db4e4838c0a603bfbd2bafa2c38ba997769043e2a6045ad9e73764b60
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b

Findings Breakdown



	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	21	0	0	0

CCR - Contract Centralization Risk

Criticality	Medium
Location	contracts/BlockSpot.sol#L184,243
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

If the `router` address is changed to an invalid implementation, then the tasks related to the router may produce unexpected behavior.

If the `lpToken` is changed, the entire dividend tracker will yield an unexpected behavior since the internal variables are accumulating the previous lptoken value.

```
function updateRouter(address newRouter) external onlyOwner {
    router = IUniswapRouter(newRouter);
}

function setLP_Token(address _lpToken) external onlyOwner {
    dividendTracker.updateLP_Token(_lpToken);
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L400,409
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
swapTokensForETH(toSwapForLiq);
swapTokensForETH(toSwapForDev);
```

Recommendation

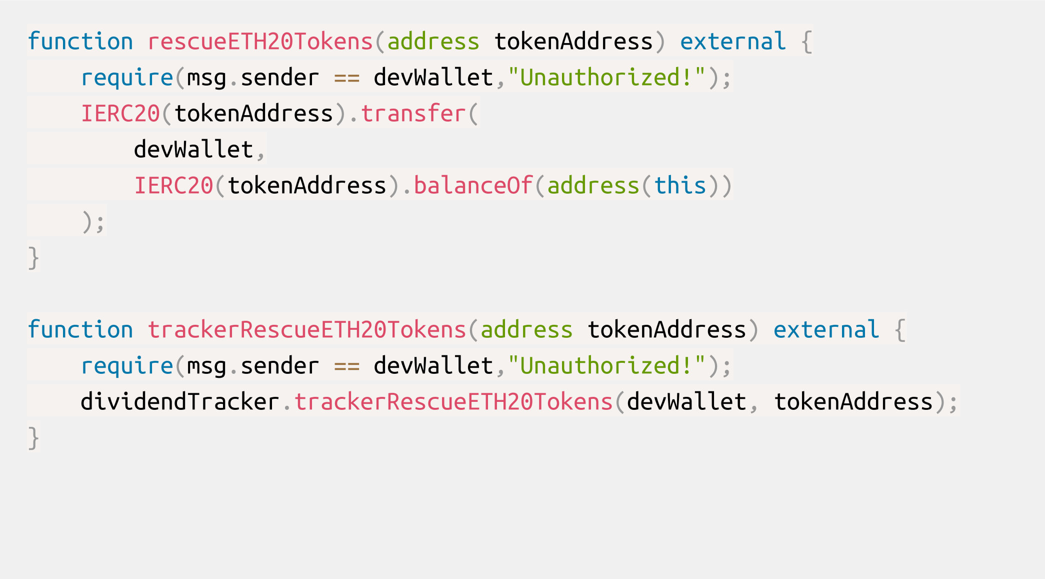
A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L162,179
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueETH20Tokens` or the `trackerRescueETH20Tokens` function.



```
function rescueETH20Tokens(address tokenAddress) external {
    require(msg.sender == devWallet,"Unauthorized!");
    IERC20(tokenAddress).transfer(
        devWallet,
        IERC20(tokenAddress).balanceOf(address(this))
    );
}

function trackerRescueETH20Tokens(address tokenAddress) external {
    require(msg.sender == devWallet,"Unauthorized!");
    dividendTracker.trackerRescueETH20Tokens(devWallet, tokenAddress);
}
```

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.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L48
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 declares some variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
mapping(address => bool) public _isBot;
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L418
Status	Unresolved

Description

The contract sends funds to a `devWallet` 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.

```
if (devAmt > 0) {
    (bool success, ) = payable(devWallet).call{value: devAmt}("");
    require(success, "Failed to send ETH to dev wallet");
}
```

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 achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L398
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 toSwapForLiq = ((tokens * sellTaxes.liquidity) / totalSellTax) /
2;
uint256 tokensToAddLiquidityWith = ((tokens * sellTaxes.liquidity) /
totalSellTax) / 2;
uint256 toSwapForDev = (tokens * sellTaxes.dev) / totalSellTax;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L396,397
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
uint256 toSwapForLiq = ((tokens * sellTaxes.liquidity) / totalSellTax) /
2;
uint256 tokensToAddLiquidityWith = ((tokens * sellTaxes.liquidity) /
totalSellTax) / 2;
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L163,173,180
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(msg.sender == devWallet,"Unauthorized!");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L176
Status	Unresolved

Description

The contract is missing error messages in `require` statements. Error messages need to accurately reflect the problem, making it easy to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(success);
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L157,211,229,233,240,514
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.

```
_isExcludedFromMaxWallet[account] = excluded;
devWallet = newWallet;
swapEnabled = _enabled;
claimEnabled = state;
_isBot[bot] = value;
LP_Token = _lpToken;
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L157,229,233
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.

```
_isExcludedFromMaxWallet[account] = excluded;
swapEnabled = _enabled;
claimEnabled = state;
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L149
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 setSwapTokensAtAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount * 10**18;
}
```

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 exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L59,61,66
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 declares certain events in its code. However, these events are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event ExcludeMultipleAccountsFromFees(address[] accounts, bool
isExcluded);
event GasForProcessingUpdated(
    uint256 indexed newValue,
    uint256 indexed oldValue
);
event ProcessedDividendTracker(
    uint256 iterations,
    uint256 claims,
    uint256 lastProcessedIndex,
    bool indexed automatic,
    uint256 gas,
    address indexed processor
);
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/LPDiv.sol
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 to 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 gas consumption unnecessarily.

```
library SafeMath {...}
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/LPDiv.sol#L25,62,68,136,143,150,160contracts/BlockSpot.sol#L48,214,220,228,243,439,513
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 public LP_Token
uint256 constant internal magnitude = 2**128
address _owner
mapping(address => bool) public _isBot
uint256 _dev
uint256 _liquidity
bool _enabled

function setLP_Token(address _lpToken) external onlyOwner {
        dividendTracker.updateLP_Token(_lpToken);
    }
address _lpToken

...
```

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

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L136,145,150,217,223
Status	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.

```
maxWallet = newNum * 10**18
maxBuyAmount = maxBuy * 10**18
swapTokensAtAmount = amount * 10**18
totalBuyTax = _liquidity + _dev
totalSellTax = _liquidity + _dev
```

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	contracts/LPDiv.sol#L170
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 _transfer(address from, address to, uint256 value) internal
virtual override {
    require(false);

    int256 _magCorrection =
magnifiedDividendPerShare.mul(value).toInt256Safe();
    magnifiedDividendCorrections[from] =
magnifiedDividendCorrections[from].add(_magCorrection);
    magnifiedDividendCorrections[to] =
magnifiedDividendCorrections[to].sub(_magCorrection);
  }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L531
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
value == true
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/BlockSpot.sol#L380,550
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.

```
uint256 feeAmt
AccountInfo memory info
```

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	contracts/BlockSpot.sol#L211,514
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.

```
devWallet = newWallet
LP_Token = _lpToken
```

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	contracts/LPDiv.sol#L3contracts/ILPDiv.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 ^0.8.10;
pragma solidity ^0.8.6;
```

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/BlockSpot.sol#L164,506
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(tokenAddress).transfer(
            devWallet,
            IERC20(tokenAddress).balanceOf(address(this))
        )

IERC20(tokenAddress).transfer(
            recipient,
            IERC20(tokenAddress).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**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		

SafeMathUint	Library			
	toInt256Safe	Internal		
IPair	Interface			
	getReserves	External		-
	token0	External		-
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IUniswapRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, Ownable		
		Public	✓	ERC20

	distributeLPDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
DividendPaying TokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
BlockSpot	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateDividendTracker	Public	✓	onlyOwner

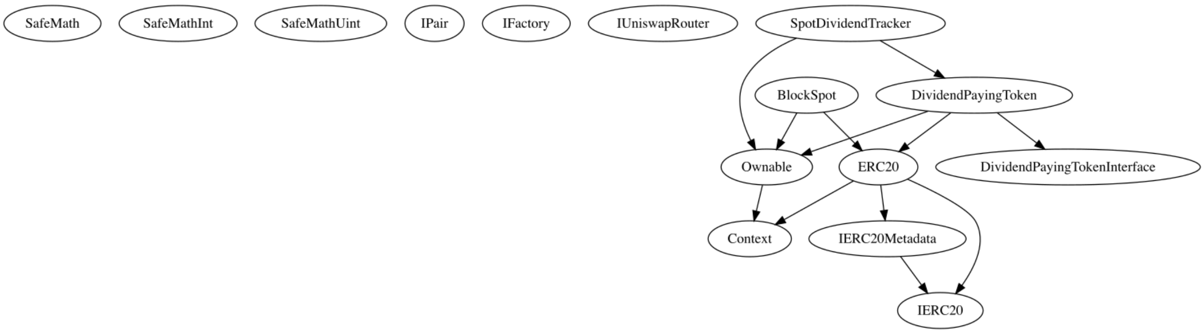
	claim	External	✓	-
	updateMaxWalletAmount	Public	✓	onlyOwner
	setMaxBuyAndSell	Public	✓	onlyOwner
	setSwapTokensAtAmount	Public	✓	onlyOwner
	excludeFromMaxWallet	Public	✓	onlyOwner
	rescueETH20Tokens	External	✓	-
	forceSend	External	✓	-
	trackerRescueETH20Tokens	External	✓	-
	updateRouter	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeFromDividends	Public	✓	onlyOwner
	setDevWallet	Public	✓	onlyOwner
	setBuyTaxes	External	✓	onlyOwner
	setSellTaxes	External	✓	onlyOwner
	setSwapEnabled	External	✓	onlyOwner
	setClaimEnabled	External	✓	onlyOwner
	setBot	External	✓	onlyOwner
	setLP_Token	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-

	dividendTokenBalanceOf	Public		-
	getAccountInfo	External		-
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	ManualLiquidityDistribution	Public	✓	onlyOwner
	swapTokensForETH	Private	✓	
	addLiquidity	Private	✓	
SpotDividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPaying Token
	trackerRescueETH20Tokens	External	✓	onlyOwner
	updateLP_Token	External	✓	onlyOwner
	_transfer	Internal		
	excludeFromDividends	External	✓	onlyOwner
	getAccount	Public		-
	setBalance	External	✓	onlyOwner
	processAccount	External	✓	onlyOwner
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			

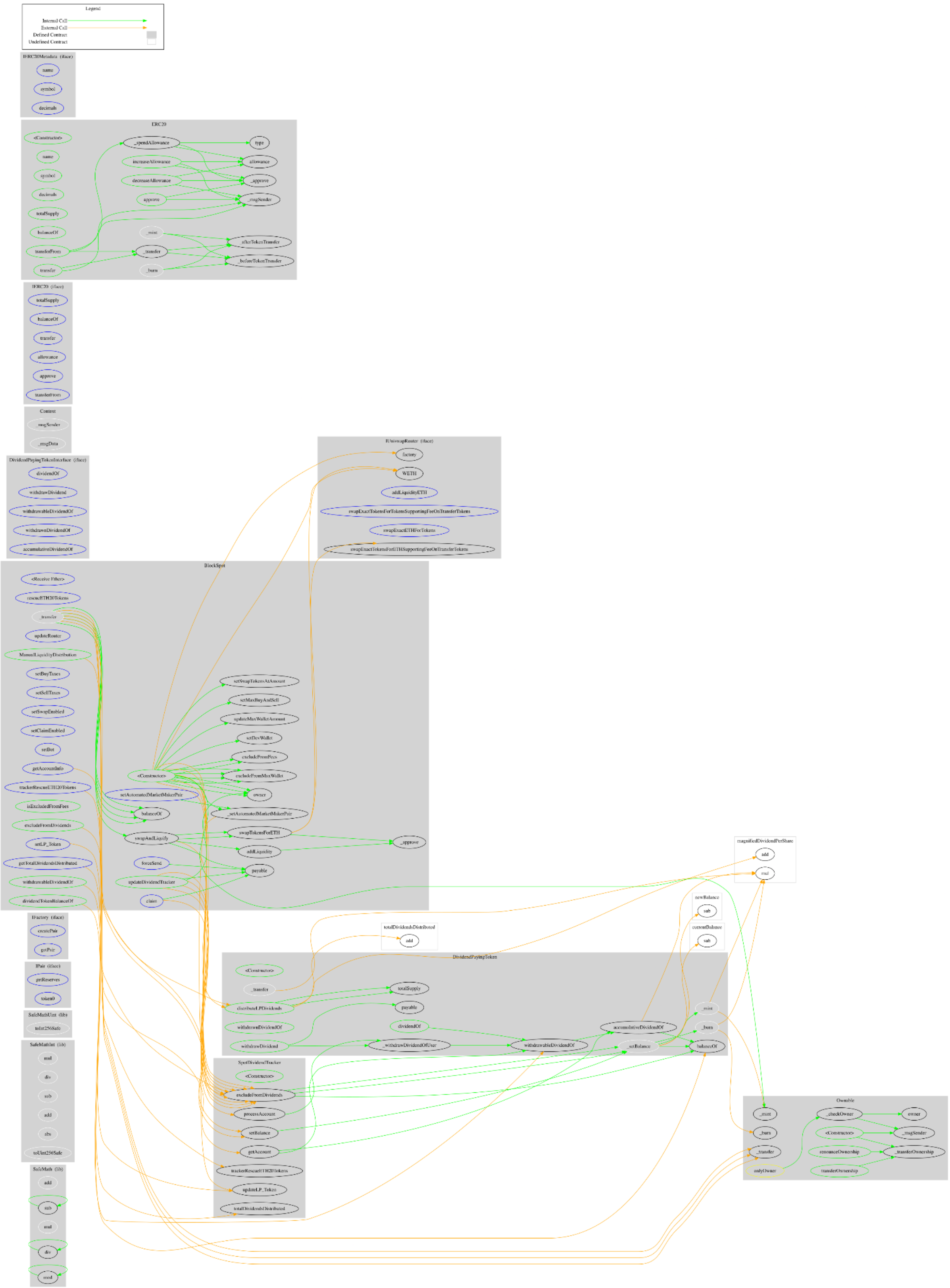
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	

	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
increaseAllowance
decreaseAllowance
approve
_mint
_transfer
_burn
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
DividendPayingTokenInterface (iface)
dividendOf
withdrawDividend
withdrawableDividendOf
withdrawnDividendOf
accumulativeDividendOf
IUniswapRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
BlockSpot
<Receive Ether>
rescueETH20Tokens
_transfer
updateRouter
ManualLiquidityDistribution
setBuyTaxes
setSellTaxes
setSwapEnabled
setClaimEnabled
setBot
getAccountInfo
trackerRescueETH20Tokens
isExcludedFromFees
excludeFromDividends
setLP_Token
getTotalDividendsDistributed
withdrawableDividendOf
dividendTokenBalanceOf
<Constructor>
setAutomatedMarketMakerPair
balanceOf
swapAndLiquify
forceSend
updateDividendTracker
claim
setSwapTokensAtAmount
setMaxBuyAndSell
updateMaxWalletAmount
setDevWallet
excludeFromFees
excludeFromMaxWallet
owner
_setAutomatedMarketMakerPair
swapTokensForETH
addLiquidity
payable
_approve
IFactory (iface)
createPair
getPair
IPair (iface)
getReserves
token0
SafeMathUint (lib)
toInt256Safe
SafeMathInt (lib)
mul
div
sub
add
abs
toUint256Safe
SafeMath (lib)
add
sub
mul
div
mod
totalDividendsDistributed
add
magnifiedDividendPerShare
add
mul
newBalance
sub
currentBalance
sub
DividendPayingToken
<Constructor>
_transfer
distributeLPDividends
withdrawnDividendOf
withdrawDividend
totalSupply
payable
dividendOf
_withdrawDividendOfUser
withdrawableDividendOf
accumulativeDividendOf
_setBalance
_mint
_burn
balanceOf
SpotDividendTracker
<Constructor>
excludeFromDividends
processAccount
setBalance
getAccount
trackerRescueETH20Tokens
updateLP_Token
totalDividendsDistributed
Ownable
_mint
_burn
_transfer
onlyOwner
_checkOwner
<Constructor>
renounceOwnership
transferOwnership
owner
_msgSender
_transferOwnership

Summary

BLOCKSPOT contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. BLOCKSPOT 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 20% buy and sell fees.

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