Cyberscope

Audit Report

ShibaWarp

September 2023

Network ETH Goerli

Address 0xAE90E7aDCDc26132568d83a5c3367CdE912763Bd

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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ZD	Zero Division	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	ShibaWarp
Compiler Version	v0.8.21+commit.d9974bed
Optimization	2000 runs
Explorer	https://goerli.etherscan.io/address/0xae90e7adcdc26132568d83a5c3367cde912763bd
Address	0xae90e7adcdc26132568d83a5c3367cde912763bd
Network	GOERLI
Symbol	SBWP
Decimals	18
Total Supply	375,000,000

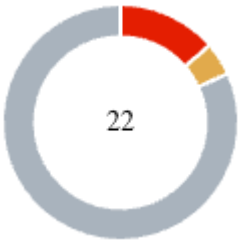
Audit Updates

Initial Audit	10 Sep 2023

Source Files

Filename	SHA256
ShibaWarp.sol	f05a1eea4df7445ff683126b855d9e2874e5ead293aa01e3a20b9761afd57408

Findings Breakdown



Critical	3
Medium	1
Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	1	0	0	0
Minor / Informative	18	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	ShibaWarp.sol#L739
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enabled the owner will not be able to disable them again.

```
if(_swapEnabled == false){
    require(_isFeeExempt[sender] || _isFeeExempt[recipient]);
}
```

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

BC - Blacklists Addresses

Criticality	Critical
Location	ShibaWarp.sol#L1018
Status	Unresolved

Description

The contract owner has the authority to stop contract addresses from transactions. The owner may take advantage of it by calling the `setBotBlacklist` function.

```
function setBotBlacklist(address _botAddress, bool _flag) external
onlyOwner {
    require(isContract(_botAddress), "only contract address, not allowed
externally owned account");
    blacklist[_botAddress] = _flag;
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ZD - Zero Division

Criticality	Critical
Location	ShibaWarp.sol#L832
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.

```
uint256 portionToDistributor = totalReceived.mul(totalRewardFee).div(totalFee);
```

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.

PAV - Pair Address Validation

Criticality	Medium
Location	ShibaWarp.sol#L840
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
address[] memory shibaPath = new address[](2);
shibaPath[0] = router.WETH();
shibaPath[1] = address(SHIBA);
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	ShibaWarp.sol#L777,852
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 variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
uint256 _totalFee = totalBuyFee;
uint256 _rewardDividendFee = rewardDividendBuyFee;
uint256 _shibaBurnFee = shibaBurnBuyFee;
uint256 _buybackFee = buybackBuyFee;
uint256 _utilityFee = utilityBuyFee;

if (recipient == pair) {
    _totalFee = totalSellFee;
    _rewardDividendFee = rewardDividendSellFee;
    _shibaBurnFee = shibaBurnSellFee;
    _buybackFee = buybackSellFee;
    _utilityFee = utilitySellFee;
}

bool distributorDepositSuccess = false;
try distributor.deposit{value: portionToDistributor}() {
    distributorDepositSuccess = true;
} catch {}
```

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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	ShibaWarp.sol#L1053
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 `rescueToken` function.

```
function rescueToken(address tokenAddress, address to) external onlyOwner
returns (bool success) {
    uint256 _contractBalance =
IERC20(tokenAddress).balanceOf(address(this));


    return IERC20(tokenAddress).transfer(to, _contractBalance);
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	ShibaWarp.sol#L1061
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `rescueBNB` method.

```
function rescueBNB(uint256 amount) external onlyOwner{
payable(msg.sender).transfer(amount);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	ShibaWarp.sol#L857,858
Status	Unresolved

Description

The contract sends funds to a `utilityReceiver` and `buybackReceiver` 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.

```
utilityReceiver.transfer(portionToUtility);
buybackReceiver.transfer(portionToBuyback);
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	ShibaWarp.sol#L932,941,1015,1020
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 modifies the state of certain variables even even when their current state matches the provided argument. As a result, the contract performs redundant storage writes.

```
_swapEnabled = true
isDividendExempt[holder] = exempt
_isFeeExempt[_addr] = true
blacklist[_botAddress] = _flag
```

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	ShibaWarp.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ShibaWarp.sol#L653,654,665,666,677
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
router
pair
distributor
DividendReceiver
_totalSupply
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ShibaWarp.sol#L342,358,591,606,612,613,616,617
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.

```
IERC20 SHIBA = IERC20(0x7C9286A16C6ab54F45CCE6b1a38Cd1ba592df895)
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
uint256 public feeDenominator = 1000
address DEAD = 0xdEAD000000000000000042069420694206942069
address ZERO = 0x0000000000000000000000000000000000000000
address public selllessSwap
uint256 public distributorDeposit
```

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	ShibaWarp.sol#L140,141,158,176,334,342,383,388,392,587,591,607,608,612,613,620,632,646,950,958,963,972,979,992,1005,1006,1014,1018,1023,1042
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address public _token
IERC20 SHIBA = IERC20(0x7C9286A16C6ab54F45CCE6b1a38Cd1ba592df895)
uint256 _minDistribution
uint256 _minPeriod
address _selllessSwap
uint256 _minSharesRequired
mapping(address => bool) _isFeeExempt
uint256 public _swapEnabledTime
uint256 public _totalSupply
address DEAD = 0xdEAD000000000000000042069420694206942069

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	ShibaWarp.sol#L7
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ShibaWarp.sol#L384,407
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.

```
minPeriod = _minPeriod
totalShares = totalShares.sub(shares[shareholder].amount).add(amount)
```

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	ShibaWarp.sol#L35
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }
```

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	ShibaWarp.sol#L735,739
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.

```
_isFeeExempt[recipient] == false
_swapEnabled == false
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ShibaWarp.sol#L791
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 feeAmount = amount.div(feeDenominator).mul(_totalFee)
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ShibaWarp.sol#L389,1008,1009
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.

```
selllessSwap = _selllessSwap
utilityReceiver = payable(_utilityReceiver)
buybackReceiver = payable(_buybackReceiver)
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	ShibaWarp.sol#L1038
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(addr) }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ShibaWarp.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 ^0.8.11;
```

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	ShibaWarp.sol#L467
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.

```
SHIBA.transfer(shareholder, amount)
```

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**
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-

	balanceOf	External		-
	allowance	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-

	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-

	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	feeTo	External		-
	feeToSetter	External		-

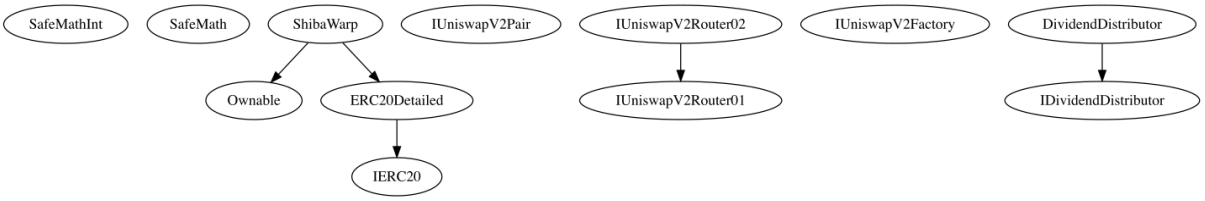
	createPair	External	✓	-
IDividendDistributor	Interface			
	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-
	setMinimumSharesRequired	External	✓	-
	setSelllessSwapAddress	External	✓	-
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
		External	Payable	-
	setDistributionCriteria	External	✓	onlyToken
	setSelllessSwapAddress	External	✓	onlyToken
	setMinimumSharesRequired	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	-
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getSelllessSwap	External		-

	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
Ownable	Implementation			
		Public	✓	-
	owner	External		-
	isOwner	Public		-
	renounceOwnership	External	✓	onlyOwner
	transferOwnership	External	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20Detailed	Implementation	IERC20		
		Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
ShibaWarp	Implementation	ERC20Detailed, Ownable		
		Public	✓	ERC20Detailed Ownable
	transfer	External	✓	-
	transferFrom	External	✓	-

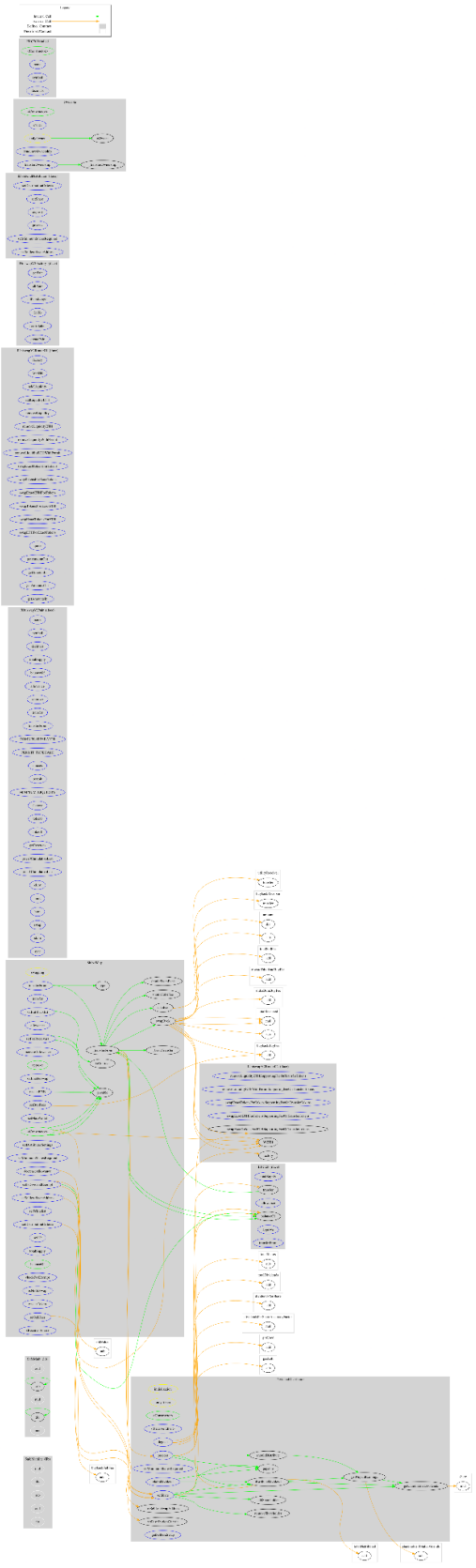
	_basicTransfer	Internal	✓	
	_transferFrom	Internal	✓	
	takeFee	Internal	✓	
	swapBack	Public	✓	swapping
	shouldTakeFee	Internal		
	shouldSwapBack	Internal		
	allowance	External		-
	decreaseAllowance	External	✓	-
	increaseAllowance	External	✓	-
	approve	Public	✓	-
	setEnableSwap	External	✓	onlyOwner
	setIsDividendExempt	External	✓	onlyOwner
	setSelllessSwapAddress	External	✓	onlyOwner
	setMaxWallet	External	✓	onlyOwner
	setDistributionCriteria	External	✓	onlyOwner
	setDistributorSettings	External	✓	onlyOwner
	setMinimumSharesRequired	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setWhitelist	External	✓	onlyOwner
	setBotBlacklist	External	✓	onlyOwner
	setLP	External	✓	onlyOwner

	totalSupply	External		-
	balanceOf	Public		-
	isContract	Internal		
	checkFeeExempt	External		-
	isNotInSwap	External		-
	rescueToken	External	✓	onlyOwner
	rescueBNB	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

ShibaWarp 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 massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% 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