Cyberscope

Audit Report

Spectre

September 2024

SHA 256:

01c1615e22e9f9f9abcae1032bdd16066b07f14200e9ed281f870a28c0263b8e

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	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	EPC	Existing Pair Creation	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PGA	Potential Griefing Attack	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSRS	Redundant SafeMath Require Statement	Unresolved
●	UTT	Unverified Token Transfer	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

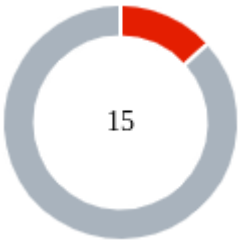
Audit Updates

Initial Audit	16 Sep 2024

Source Files

Filename	SHA256
SpectreForAudit.sol	01c1615e22e9f9f9abcae1032bdd16066b07f14200e9ed281f870a28c0263b8e

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

BC - Blacklists Addresses

Criticality	Critical
Location	SpectreForAudit.sol#L395
Status	Unresolved

Description

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

```
function addBots(address[] memory bots_) public onlyOwner {
        for (uint256 i = 0; i < bots_.length; i++) {
            bots[bots_[i]] = true;
        }
}
```

```
function _transfer(address from,address to,uint256 amount)
private {
    ...
        if (from != owner() && to != owner() &&
        from != _spectreMultiSegWallet && to !=
_spectreMultiSegWallet) {
            require(!bots[from] && !bots[to], "Transaction
blocked: Address is flagged as a bot.");
        ...
        }
    ...
```

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.

EPC - Existing Pair Creation

Criticality	Critical
Location	SpectreForAudit.sol#L411
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert. In this case, the contract will be unable to deposit its token balance as liquidity or swap accrued fees for ETH.

```
function openTrading() external onlyOwner {
        require(!tradingOpen, "trading is already open");
        uniswapV2Router = IUniswapV2Router02(
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        _approve(address(this), address(uniswapV2Router), _tTotal);
        uniswapV2Pair =
IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(this),uni
swapV2Router.WETH());
        uniswapV2Router.addLiquidityETH{value: address(this).balance}(
            address(this),
            balanceOf(address(this)),
            0,
            0,
            owner(),
            block.timestamp
        );
        IERC20(uniswapV2Pair).approve(
            address(uniswapV2Router),
            type(uint256).max
        );
        swapEnabled = true;
        tradingOpen = true;
    }
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before proceeding to create a new pair. This can be achieved by utilizing the getPair function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the getPair function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the createPair function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the createPair function will revert.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L176,179,180,181,182,205,207,212,307,312,320,330,335,358,360,
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
if (taxAmount > 0) {
    _balances[address(this)] =
_balances[address(this)].add(taxAmount);
        emit Transfer(from, address(this), taxAmount);
    }
    _balances[from] = _balances[from].sub(amount);
    _balances[to] = _balances[to].add(amount.sub(taxAmount));
    emit Transfer(from, to, amount.sub(taxAmount));
```

```
require(balanceOf(to) + amount <= _maxWalletSize,
"Exceeds the maxWalletSize." );
if (_buyCount <= (_reduceBuyTaxAt + _reduceSellTaxAt)) {...}
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L381,395,401,411,467
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.

```
function updateSpectreTaxWallet( address payable
_newSpectreMultiSegWallet
        ) external onlyOwner {}

function removeSpectreLimits() external onlyOwner {}

function addBots(address[] memory bots_) public onlyOwner {}

function delBots(address[] memory notbot) public onlyOwner {}

function openTrading() external onlyOwner {}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L395,401,437,467
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 addBots(address[] memory bots_) public onlyOwner {
        for (uint256 i = 0; i < bots_.length; i++) {
            bots[bots_[i]] = true;
        }
    }
```

```
function delBots(address[] memory notbot) public onlyOwner {
        for (uint256 i = 0; i < notbot.length; i++) {
            bots[notbot[i]] = false;
        }
}
```

```
function reduceSpectreSellTaxFee(uint256 _newFee) external
onlySpectreWallet {
        require(_newFee <= _finalSellTax, "Invalid fee: New fee
must be lower or equal to current sell tax.");
        _finalSellTax = _newFee;
}
```

```
function updateSpectreTaxWallet(address payable
_newSpectreMultiSegWallet) external onlyOwner {
        require(_newSpectreMultiSegWallet != address(0), "Enter
a valid wallet");
        _spectreMultiSegWallet =
payable(_newSpectreMultiSegWallet);
}
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L306
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. Misleading variable names can lead to confusion, making the code more difficult to read and understand. In this case, the variables `_finalBuyTax` and `_initialBuyTax` are particularly misleading, as their names suggest they are intended to apply only to buy transactions. However, these variables are used to impose taxation on every transaction, irrespectively of its nature.

```
function _transfer(address from,address to,uint256 amount)
private {
    ...
    if (from != owner() && to != owner() && from !=
_spectreMultiSegWallet && to != _spectreMultiSegWallet) {
        require(!bots[from] && !bots[to], "Transaction blocked:
Address is flagged as a bot.");
        taxAmount = amount.
        mul((_buyCount > _reduceBuyTaxAt? _finalBuyTax:
_initialBuyTax).div(100);
    ...
    }
...
}
```

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.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L302
Status	Unresolved

Description

The contract is not fully conforming to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However the contract implements, a conditional check that prohibits transfers of 0 values.

This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address from,address to,int256 amount)
private {
...
require(amount > 0, "Transfer amount must be greater than
zero");
...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L343
Status	Unresolved

Description

The `_transfer` function includes a `require` statement intended to impose a maximum limit of 2 sales in a single block. This opens up the potential for a griefing attack. During such an incident, a malicious actor could front-run normal transactions with spam transactions to prevent legitimate sales from taking place.

```
function _transfer(address from,address to,uint256 amount) private {
...
if (!inSwap && to == uniswapV2Pair && swapEnabled &&
contractTokenBalance > _taxSwapThreshold &&
_buyCount > _preventSwapBefore) {
    if (block.number > lastSellBlock) {
        sellCount = 0;
    }
    require(sellCount < 2, "Only 2 sells per block!");
    swapTokensForEth(min(amount, min(contractTokenBalance,
_maxTaxSwap)));
...
}
```

Recommendation

The team is advised to review the transfer mechanism in regards to sales to ensure all legitimate transactions are processed according to the intended behavior. A lower threshold on the transferred amount could be implemented before the sales counter is incremented.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L367
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 swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router),
tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
            tokenAmount,
            0,
            path,
            address(this),
            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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L347,388
Status	Unresolved

Description

The contract sends funds to a `_spectreMultiSegWallet` 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.

```
function sendETHToMultiSeg(uint256 amount) private {
        (bool success, ) = _spectreMultiSegWallet.call{value:
amount}("");
        if (!success) {
            _spectreWallet.transfer(amount);
        }
}
```

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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L367
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router),
tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
}
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SpectreForAudit.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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L48
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

UTT - Unverified Token Transfer

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L442
Status	Unresolved

Description

The contract transfers tokens implementing the standard transfer method. If the transfer fails due to insufficient funds, contract issues, or other reasons, the transaction does not automatically revert, potentially leading the contract to falsely signal success without confirming the transfer occurred.

```
function sendTokensToSpectreMultiSeg(address tokenAddress)
        external onlySpectreWallet
        returns (bool success)
{
        emit clearTokens(tokenAddress,
IERC20(tokenAddress).balanceOf(address(this)));
        return IERC20(tokenAddress).transfer(_spectreMultiSegWallet,
IERC20(tokenAddress).balanceOf(address(this)));
}
```

Recommendation

The team is advised to monitor the success of the token transfer and revert the transaction if the transfer fails. By implementing such an approach, it is ensured that the function will not silently succeed in case of an unsuccessful transfer, providing a more secure and reliable token transfer process.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SpectreForAudit.sol#L137,164,166,167,169,170,171,172,175,176,177,178,179,180,193,437,467
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 payable public _spectreMultiSegWallet
uint256 private constant _initialBuyTax = 20
uint256 private constant _initialSellTax = 20
uint256 public _finalSellTax = 2
uint256 private constant _reduceBuyTaxAt = 20
uint256 private constant _reduceSellTaxAt = 20
uint256 private constant _preventSwapBefore = 20
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 100000000000 * 10**_decimals
string private constant _name = unicode"Spectre"
string private constant _symbol = unicode"SPEC"
uint256 public _maxTxAmount = _tTotal * 1 / 100
uint256 public _maxWalletSize = _tTotal * 1 / 100

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

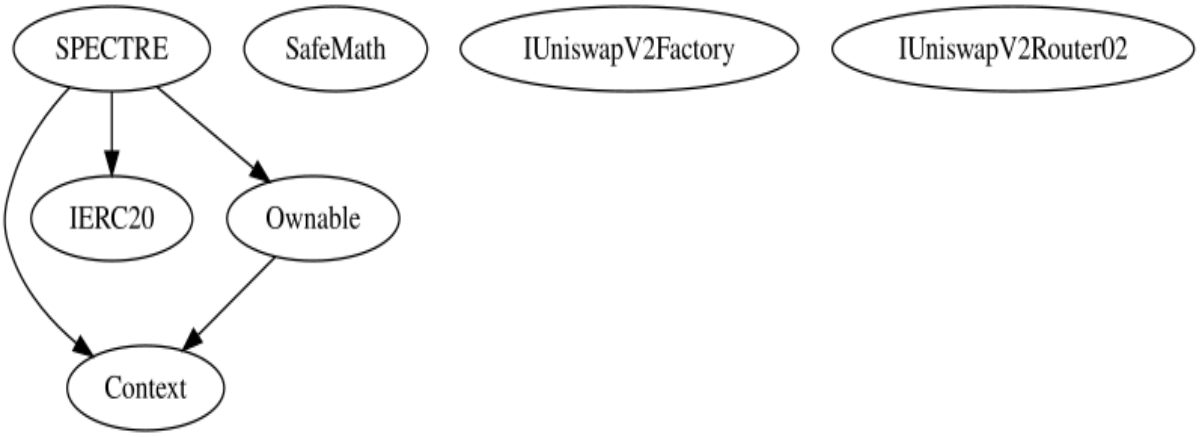
Find more information on the Solidity documentation https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

Functions Analysis

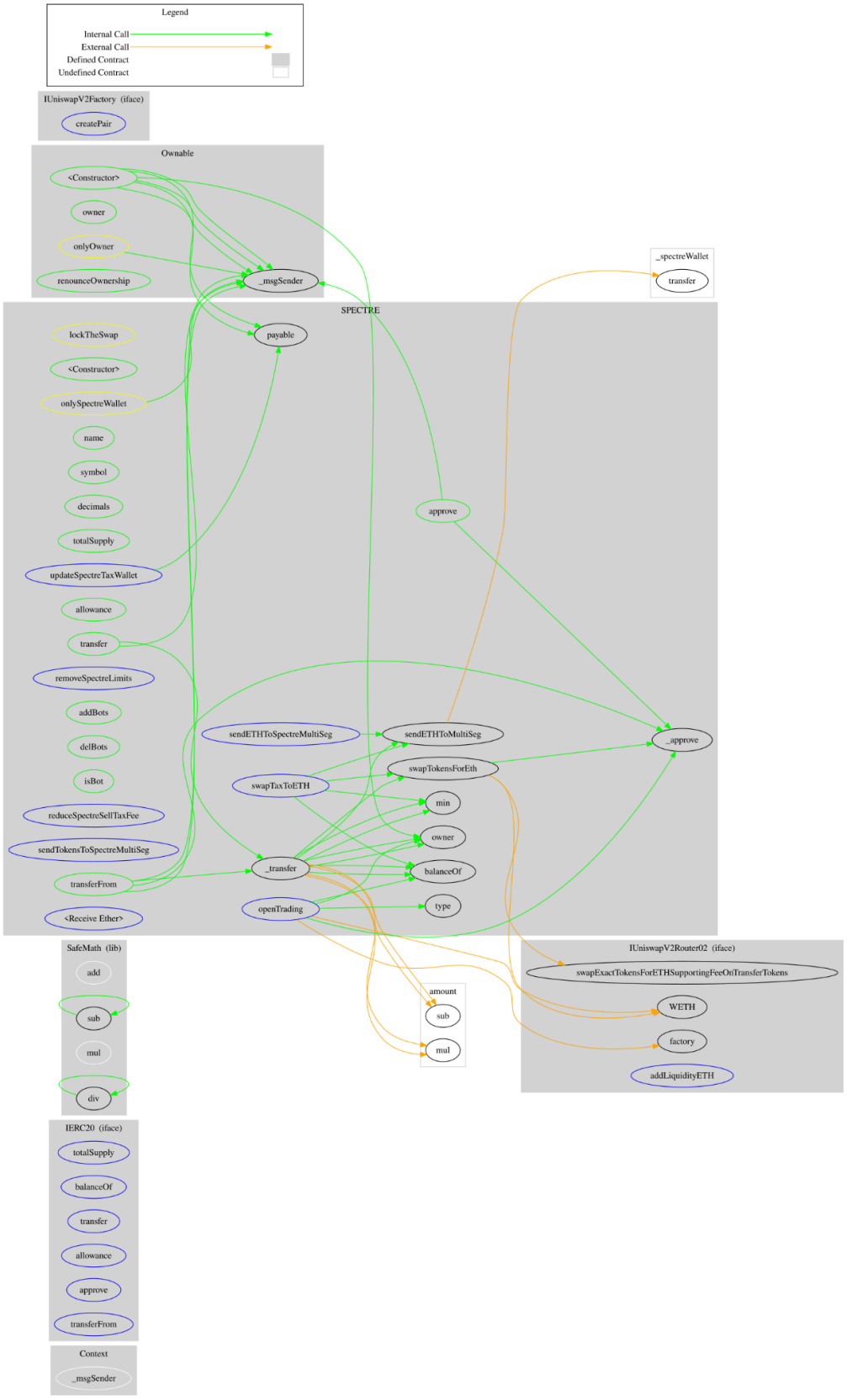
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SPECTRE	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeSpectreLimits	External	✓	onlyOwner
	sendETHToMultiSeg	Private	✓	
	addBots	Public	✓	onlyOwner
	delBots	Public	✓	onlyOwner
	isBot	Public		-
	openTrading	External	✓	onlyOwner

	reduceSpectreSellTaxFee	External	✓	onlySpectreWallet
	sendTokensToSpectreMultiSeg	External	✓	onlySpectreWallet
	swapTaxToETH	External	✓	onlySpectreWallet
	sendETHToSpectreMultiSeg	External	✓	onlySpectreWallet
	updateSpectreTaxWallet	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
_msgSender
_spectreWallet
transfer
SPECTRE
lockTheSwap
payable
<Constructor>
onlySpectreWallet
name
symbol
decimals
totalSupply
updateSpectreTaxWallet
allowance
transfer
removeSpectreLimits
addBots
delBots
isBot
reduceSpectreSellTaxFee
sendTokensToSpectreMultiSeg
transferFrom
<Receive Ether>
approve
sendETHToSpectreMultiSeg
sendETHToMultiSeg
_approve
swapTokensForEth
swapTaxToETH
min
owner
_transfer
balanceOf
openTrading
type
amount
sub
mul
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
SafeMath (lib)
add
sub
mul
div
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

SPECTRE is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error and two critical issues. The contract Owner can access some admin functions that can be used in a malicious way to disturb the users' transactions. Other issues of minor severity were identified to improve the contract's performance and consistency.

This audit investigated security issues, business logic concerns and potential improvements.

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

cyberscope.io