Cyberscope

Audit Report

SocialPal

April 2024

SHA256 36c3569ee023af7bdd739ed80c03ab9e212ecc829f666f2c4fb71099d5d65aa5

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	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	DDP	Decimal Division Precision	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RCV	Redundant Calculation Value	Unresolved
	RSD	Redundant Swap Duplication	Unresolved
	RC	Repetitive Calculations	Unresolved
	UTA	Unnecessary Token Approval	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L18	Multiple Pragma Directives	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	SPL
Testing Deploy	https://testnet.bscscan.com/address/0xdce423009c59de269935492e22d6a54cb0432af7
Symbol	SPL
Decimals	18
Total Supply	140,000,000
Badge Eligibility	Yes

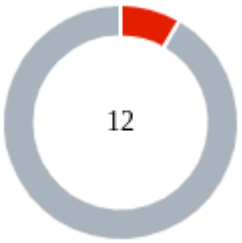
Audit Updates

Initial Audit	24 Apr 2024 https://github.com/cyberscope-io/audits/blob/main/3-spl/v1/audit.pdf
Corrected Phase 2	26 Apr 2024

Source Files

Filename	SHA256
contracts/SPL.sol	36c3569ee023af7bdd739ed80c03ab9e212ecc829f666f2c4fb71099d5d65aa5

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/SPL.sol#L577
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 enable the owner will not be able to disable them again.

```
if (isLimitedAddress(from, to)) {
    require(isTradingEnabled, "Trading is not enabled");
}
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/SPL.sol#L583,588
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.

```
if (devAllocation > 0) {
    internalSwap((contractTokenBalance * devAllocation) / 100);
}

if (liquidityAllocation > 0) {
    swapAndLiquify(
        (contractTokenBalance * liquidityAllocation) / 100
    );
}
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/SPL.sol#L583,771
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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.

```
if (contractTokenBalance >= swapThreshold) {
    if (devAllocation > 0) {
        internalSwap((contractTokenBalance * devAllocation) /
100);
    }

    if (liquidityAllocation > 0) {
        swapAndLiquify(
            (contractTokenBalance * liquidityAllocation) / 100
        );
    }
...
function triggerInternalSwap() external onlyOwner {
    uint256 contractTokenBalance = balanceOf(address(this));
    ...
    internalSwap((contractTokenBalance * devAllocation) / 100);
}
```

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.

RCV - Redundant Calculation Value

Criticality	Minor / Informative
Location	contracts/SPL.sol#L2383
Status	Unresolved

Description

The contract is using the `triggerInternalSwap` function that is designed to handle internal swaps based on a developer allocation (`devAllocation`). This function calculates the portion of tokens to be swapped by using the current contract's token balance and the `devAllocation` percentage. However, since `triggerInternalSwap` can be invoked multiple times without restrictions other than ownership and token balance, the intended control over the swapping process by using `devAllocation` does not effectively limit or manage the frequency or total volume of tokens being swapped. This redundancy in functionality could lead to potential misuse or unintended financial implications.

```
function triggerInternalSwap() external onlyOwner {
    uint256 contractTokenBalance = balanceOf(address(this));
    require(contractTokenBalance > 0, "NO_FEE_BAL");
    require(devAllocation > 0, "INVALID_DEV_ALLOC");
    internalSwap((contractTokenBalance * devAllocation) / 100);
}
```

Recommendation

It is recommended to introduce a mechanism to limit or track the invocation of `triggerInternalSwap` to ensure that the developer allocation is used as intended and not exploited by repeated calls. Options could include implementing a cooldown period between swaps, tracking the cumulative amount swapped within a certain timeframe against a predefined limit, or requiring a dynamic condition based on other contract states or external factors. These measures would help maintain the integrity of the `devAllocation` strategy and prevent potential abuse of the swapping functionality, thereby aligning the actual functionality with the intended financial and operational controls.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/SPL.sol#L582
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.

```
if (devAllocation > 0) {
    internalSwap((contractTokenBalance * devAllocation) / 100);
}

if (liquidityAllocation > 0) {
    swapAndLiquify(
        (contractTokenBalance * liquidityAllocation) / 100
    );
}
```

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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/SPL.sol#L579,599,2375
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.

```
    function _transfer(

        ...

        if (is_sell(from, to) && !inSwap && canSwap(from, to))
{
            ...
        uint256 amountAfterFee = (takeFee)
            ? takeTaxes(from, is_buy(from, to), is_sell(from,
to), amount)
        ...
        require(
            _threshold * (10 ** _decimals) <= _totalSupply,
            "INVALID_THRESHOLD"
        );
        swapThreshold = _threshold * (10 ** _decimals);
```

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

UTA - Unnecessary Token Approval

Criticality	Minor / Informative
Location	contracts/SPL.sol#L468
Status	Unresolved

Description

It was identified that the contract's constructor grants an infinite approval to the Uniswap router for both the contract creator and the contract itself. This approval is granted through the `_approve` function with the maximum possible allowance value, `type(uint256).max`, effectvely allowing the Uniswap router to spend tokens from the contract creator without limit.

```
    constructor(address _devWallet) Ownable(address(msg.sender))
{
        ...

        if (block.chainid == 56) {
            swapRouter =
IRouter02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        } else if (block.chainid == 97) {
            swapRouter =
IRouter02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);
        } else if (
            block.chainid == 1 || block.chainid == 4 ||
block.chainid == 3
        ) {
            swapRouter =
IRouter02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        } else if (block.chainid == 42161) {
            swapRouter =
IRouter02(0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506);
        } else if (block.chainid == 5) {
            swapRouter =
IRouter02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        } else {
            revert("Chain not valid");
        }

        ...
        _approve(msg.sender, address(swapRouter),
type(uint256).max);
        _approve(address(this), address(swapRouter),
type(uint256).max);
    }
```

Recommendation

It is advised to review the necessity of granting infinite approval to the Uniswap router during contract deployment. If this approval is not essential for the contract's intended functionality, it should be removed or limited to the necessary allowance amount required for specific operations.

By restricting approval to the minimum necessary level, the contract can minimize the potential impact of malicious activities, such as unauthorized token transfers or manipulative trading behaviors, while still maintaining functionality with external protocols like Uniswap.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/SPL.sol#L276,392,409,410,422,423,424,425,426,427,428,429,430,431,432,433,546,551,756,757,765
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);
uint8 private constant _decimals = 18
string private constant _name = "SocialPal"
string private constant _symbol = "SPL"
event _enableTrading();
event _setPresaleAddress(address account, bool enabled);
event _toggleCanSwapFees(bool enabled);
event _changeThreshold(uint256 newThreshold);
event _changeDevWallet(address newSell);
event _changeFees(uint256 buy, uint256 sell, uint256 transfer);
event _setStakingAddress(address staking);
event _swapAndLiquify();
event _changeFeeAllocations(uint256 dev, uint256 liquify);
event _internalSwap();

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/SPL.sol#L31
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 _contextSuffixLength() internal view virtual returns
(uint256) {
        return 0;
    }
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/SPL.sol#L724
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool success
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/SPL.sol#L10,41,145,241
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.20;
pragma solidity ^0.8.24;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/SPL.sol#L10,41,145,241
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 ^0.8.20;
pragma solidity ^0.8.24;
```

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.

Functions Analysis

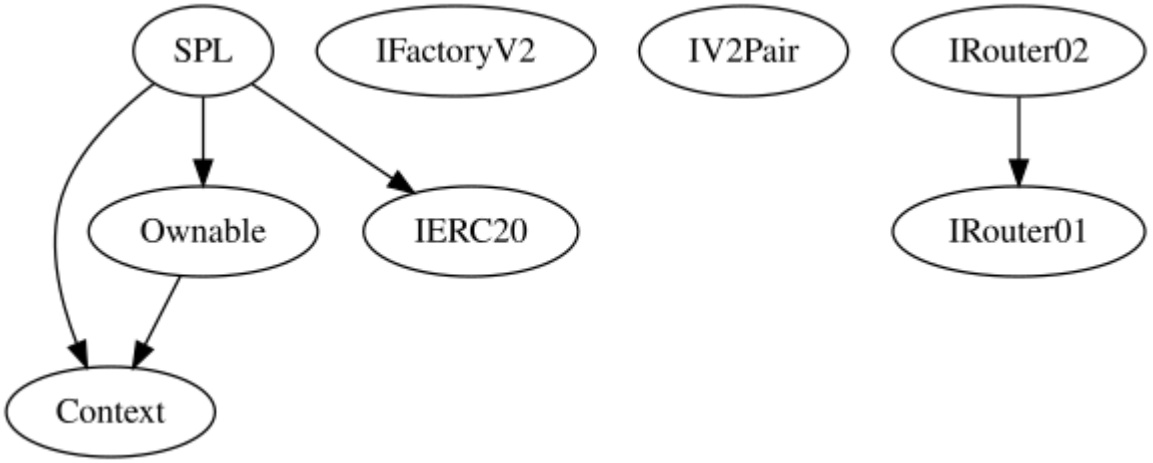
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

	transferFrom	External	✓	-
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-
	getReserves	External		-
	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-

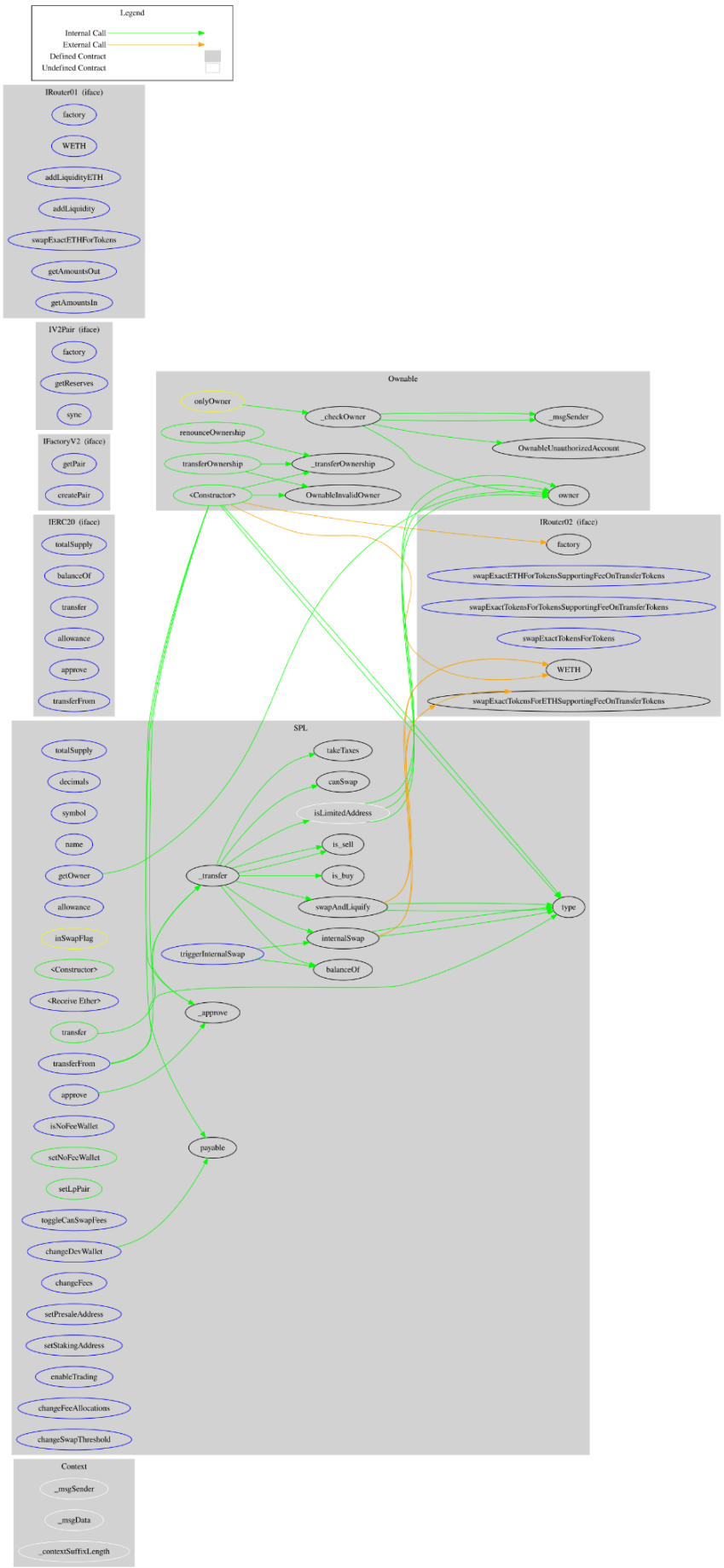
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
SPL	Implementation	Context, Ownable, IERC20		
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
		Public	✓	Ownable
		External	Payable	-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	transferFrom	External	✓	-
	isNoFeeWallet	External		-
	setNoFeeWallet	Public	✓	onlyOwner
	setLpPair	Public	✓	onlyOwner
	isLimitedAddress	Internal		
	is_buy	Internal		
	is_sell	Internal		

	canSwap	Internal		
	toggleCanSwapFees	External	✓	onlyOwner
	_transfer	Internal	✓	
	changeDevWallet	External	✓	onlyOwner
	changeFees	External	✓	onlyOwner
	takeTaxes	Internal	✓	
	swapAndLiquify	Internal	✓	inSwapFlag
	internalSwap	Internal	✓	inSwapFlag
	setPresaleAddress	External	✓	onlyOwner
	setStakingAddress	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	changeFeeAllocations	External	✓	onlyOwner
	changeSwapThreshold	External	✓	onlyOwner
	triggerInternalSwap	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
_msgSender
renounceOwnership
OwnableUnauthorizedAccount
transferOwnership
_transferOwnership
<Constructor>
OwnableInvalidOwner
owner
IRouter02 (iface)
factory
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
SPL
totalSupply
decimals
symbol
name
getOwner
allowance
inSwapFlag
<Constructor>
<Receive Ether>
transfer
transferFrom
approve
isNoFeeWallet
setNoFeeWallet
setLpPair
toggleCanSwapFees
changeDevWallet
changeFees
setPresaleAddress
setStakingAddress
enableTrading
changeFeeAllocations
changeSwapThreshold
takeTaxes
canSwap
isLimitedAddress
is_sell
is_buy
_transfer
swapAndLiquify
type
internalSwap
triggerInternalSwap
balanceOf
_approve
payable
Context
_msgSender
_msgData
_contextSuffixLength

Summary

SocialPal 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. 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