Cyberscope

Audit Report

Swytch Token

February 2024

Network	BSC
Address	0x362c108b7015f7676b603f6a67de4ca3a29639ec
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
🔴	ZD	Zero Division	Unresolved
⚪	IDI	Immutable Declaration Improvement	Unresolved
⚪	PTRP	Potential Transfer Revert Propagation	Unresolved
⚪	PVC	Price Volatility Concern	Unresolved
⚪	RLC	Redundant Launch Check	Unresolved
⚪	RSW	Redundant Storage Writes	Unresolved
⚪	SPI	Swap Parameter Inconsistent	Unresolved
⚪	OCTD	Transfers Contract's Tokens	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L09	Dead Code Elimination	Unresolved
⚪	L15	Local Scope Variable Shadowing	Unresolved
⚪	L16	Validate Variable Setters	Unresolved

●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	TOKEN_CONTRACT
Compiler Version	v0.8.16+commit.07a7930e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x362c108b7015f7676b603f6a67de4ca3a29639ec
Address	0x362c108b7015f7676b603f6a67de4ca3a29639ec
Network	BSC
Symbol	Swytch
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

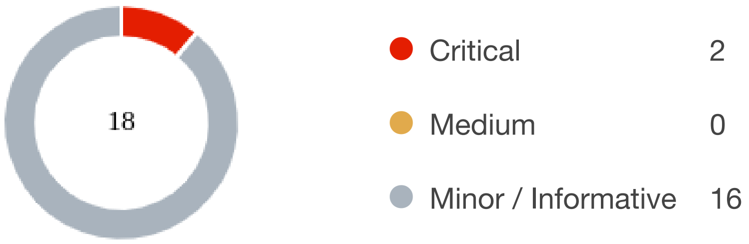
Audit Updates

Initial Audit	28 Feb 2024

Source Files

Filename	SHA256
TOKEN_CONTRACT.sol	a69b0d6b93e78349d167e80f2791f327c1a409ce842383569328f6b7439716c1

Findings Breakdown



	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	16	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	TOKEN_CONTRACT.sol#L1507,1510
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users, as described in detail in sections `ZD` and `PTRP`. As a result, the contract might operate as a honeypot.

```
uint256 amountOfTokenMarketing = (taxAmount * marketingFee) /
                (totalFee);
uint256 amountOfTokenTx = (taxAmount * txFee) / (totalFee);
_transfer(address(this), marketingWallet,
amountOfTokenMarketing);
_transfer(address(this), devWallet, amountOfTokenTx);
```

Recommendation

It is recommended to address the findings identified in the `ZD` and `PTRP` sections, where the contract owner possesses the unilateral authority to halt sales for all users. 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.

ZD - Zero Division

Criticality	Critical
Location	TOKEN_CONTRACT.sol#L1507
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert. Specifically the `totalFee` which is used as denominator, can be set to zero.

```
uint256 amountOfTokenMarketing = (taxAmount * marketingFee) /
                (totalFee);
uint256 amountOfTokenTx = (taxAmount * txFee) / (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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1414,1415
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.

```
launchedAt
launchedAtTimestamp
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1510
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` and `devWallet` addresses as part of the transfer flow. These address can either be set to the zero address. If the addressese is set to a zero address then it well revert the transaction. As a result, the error will propagate to the token's contract and revert the transfer.

```
_transfer(address(this), marketingWallet,
amountOfTokenMarketing);
_transfer(address(this), devWallet, amountOfTokenTx);
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	1497,TOKEN_CONTRACT.sol#L1472,1497,1504
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapEnabled` sets a flag when the contract will trigger the swap functionality. If the variable is set to false to a long time, then the contract will accumulate tokens to swap. When the `swapEnabled` is toggle to `true` 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 (shouldSwapBack()) {
            swapBack();
        }

    function shouldSwapBack() internal view returns (bool) {
        return
            !inSwap &&
            swapEnabled &&
            launched() &&
            balanceOf(address(this)) > 0 &&
            !isPair(_msgSender());
    }

    function swapBack() internal isSwapping {
        uint256 taxAmount = balanceOf(address(this));
        _approve(address(this), address(swapRouter),
taxAmount);

        ...

        if (addLiquidityEnabled) {
            _canAddLp();
        }
    }
```

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.

RLC - Redundant Launch Check

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1417,1581
Status	Unresolved

Description

The contract contains the `launched` function to determine if the trading functionalities should be enabled. Specifically, the `launchedAt` variable is checked to ensure it is not equal to zero, with the intent of verifying that the contract has been launched. However, since `launchedAt` is initialized within the contract's constructor, this condition will always evaluate to `true` after the contract's deployment, rendering the check ineffective. This oversight means that the intended gatekeeping logic, which relies on the state of `launchedAt` to enable or disable trading and liquidity additions, fails to serve its purpose as it does not accurately reflect the contract's operational status post-launch.

```
    constructor(
        ...
    ) payable ERC20(_tokenName, _tokenSymbol, _decimals) {
        ...

        launchedAt = block.number;
        ...

    function launched() internal view returns (bool) {
        return launchedAt != 0;
    }
```

Recommendation

It is recommended to reassess the contract's launch logic to ensure that the `launchedAt` variable serves its intended purpose effectively. If `launchedAt` is meant to act as a switch for certain functionalities, consider implementing an explicit initialization phase where `launchedAt` is set to zero and can be updated to a non-zero value upon an actual launch event. This approach would necessitate adding an administrative function to update the `launchedAt` state under specific conditions, preferably protected by appropriate access control mechanisms to prevent unauthorized manipulation. Otherwise conside to remoce the `launched` functionality from the codebase.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1683
Status	Unresolved

Description

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

```
    function setIsFeeExempt(address holder, bool exempt)
external onlyOwner {
        isExemptedFromFee[holder] = exempt;
    }
```

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.

SPI - Swap Parameter Inconsistent

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1529
Status	Unresolved

Description

The contract is utilizing the `swapExactTokensForETHSupportingFeeOnTransferTokens` function for swapping tokens for ETH. However, in its first execution attempt within a try-catch block, an erroneous and redundant `address(0)` parameter is included. This parameter does not align with the expected signature of the swap function, which typically requires the amount of tokens to swap, the minimum amount of ETH to accept, the path for the swap, the recipient address, and the transaction deadline. The inclusion of `address(0)` as an extra parameter may lead to execution failure or, in some contexts, a compilation error, as it does not match the function signature defined in standard swap router interfaces. This oversight could potentially prevent the function from executing as intended, impacting the contract's ability to swap tokens efficiently and securely.

```
       try
swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
                half,
                0,
                pathEth,
                address(this),
                address(0),
                block.timestamp
            )
        {
            success = true;
        } catch {
            try
swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
                    half,
                    0,
                    pathEth,
                    address(this),
                    block.timestamp
                )
            {
                success = true;
            } catch {}
```

Recommendation

It is recommended to remove the redundant `address(0)` parameter from the first `swapExactTokensForETHSupportingFeeOnTransferTokens` function call in order to align with the expected function interface. Ensuring that the function call correctly matches the swap router's API will eliminate the risk of execution failure due to signature mismatch. Additionally, a thorough review of the contract should be conducted to identify and correct any similar issues with function calls.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1608
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) external
onlyOwner {
        IERC20(tokenAddress).safeTransfer(
            msg.sender,
            IERC20(tokenAddress).balanceOf(address(this))
        );
    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1347
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.

```
uint256 public feeDenominator = 10000
```

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	TOKEN_CONTRACT.sol#L351,1117,1301,1324,1437,1648,1649,1650,1660,1661,1662,1672,1684,1688
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 WETH() external pure returns (address);

...
```

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	TOKEN_CONTRACT.sol#L1653,1665,1684
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.

```
liquidityFeeBuy = _liquidityFee
liquidityFeeSell = _liquidityFee
isExemptedFromFee[holder] = exempt;
```

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	TOKEN_CONTRACT.sol#L264,368,381,402,438,451,466,478,523,569,581,599,615,638,844,873,883,894,898,914,973,1011,1021,1032,1036,1052
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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

...
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1368,1370
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint8 _decimals
uint256 _totalSupply
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L1378,1404,1405,1676,1677
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.

```
(bool success, ) = payable(_devWallet).call{value:
msg.value}("")
devWallet = _admin
marketingWallet = _marketingWallet
devWallet = _devWallet
```

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	TOKEN_CONTRACT.sol#L543,923,982,1061
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
            result := store
        }
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	TOKEN_CONTRACT.sol#L4,41,65,77,324,354,553,676,734,1069,1112,1221,1278,1299
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 >=0.5.0;
pragma solidity >=0.6.2;
pragma solidity ^0.8.0;
pragma solidity ^0.8.1;
pragma solidity ^0.8.16;
```

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	TOKEN_CONTRACT.sol#L4,41,65,77,324,354,553,676,734,1299
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 ^0.8.0;
pragma solidity ^0.8.1;
pragma solidity ^0.8.16;
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
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	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-

	DOMAIN_SEPARATOR	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	

	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
EnumerableSet	Library			
	_add	Private	✓	
	_remove	Private	✓	
	_contains	Private		
	_length	Private		
	_at	Private		
	_values	Private		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		

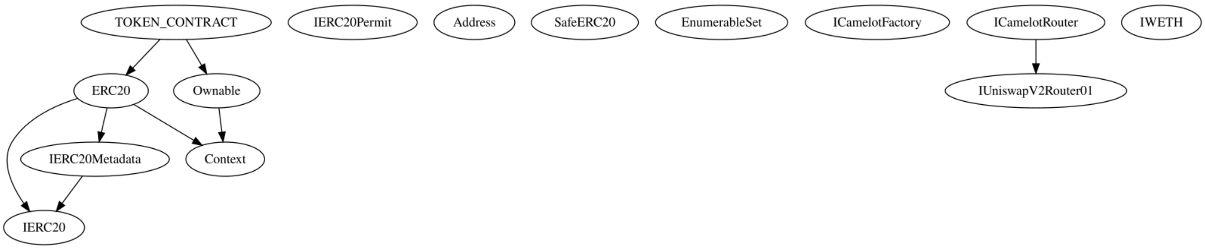
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
ICamelotFactory	Interface			
	owner	External		-
	feePercentOwner	External		-
	setStableOwner	External		-
	feeTo	External		-
	ownerFeeShare	External		-
	referrersFeeShare	External		-
	getPair	External		-
	allPairs	External		-

	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	feeInfo	External		-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	quote	External		-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
ICamelotRouter	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

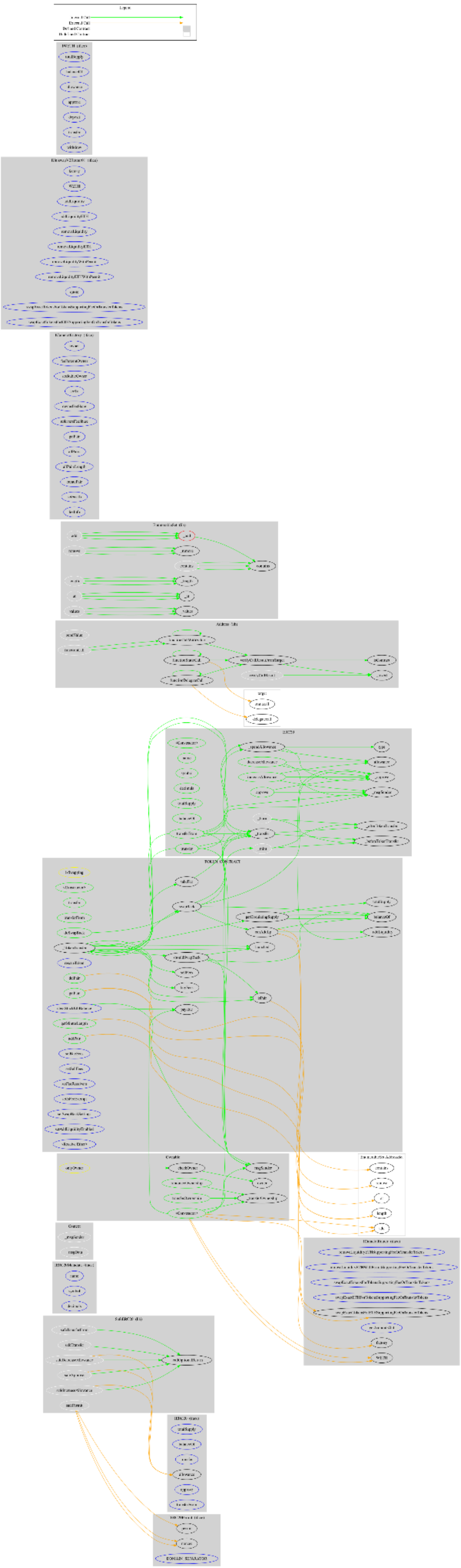
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	getAmountsOut	External		-
IWETH	Interface			
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	deposit	External	Payable	-
	transfer	External	✓	-
	withdraw	External	✓	-
TOKEN_CONTRACT	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_TokenTransfer	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	isSwapping
	_canAddLp	Internal	✓	
	_addLiquidity	Internal	✓	
	doSwapBack	Public	✓	onlyOwner

	launched	Internal		
	buyFees	Internal	✓	
	sellFees	Internal	✓	
	takeFee	Internal	✓	
	rescueToken	External	✓	onlyOwner
	clearStuckEthBalance	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	isPair	Public		-
	addPair	Public	✓	onlyOwner
	delPair	Public	✓	onlyOwner
	getMinterLength	Public		-
	getPair	Public		-
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setIsFeeExempt	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setAddLiquidityEnabled	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


functionCallWithValue
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
decreaseAllowance
increaseAllowance
approve
_burn
_transfer
_mint
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
TOKEN_CONTRACT
takeFee
swapBack
getCirculatingSupply
totalSupply
balanceOf
addLiquidity

Summary

Swytch Token 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