Cyberscope

Audit Report

Marvin Token

June 2024

SHA256 a016e00d69d68d32c37de47b0478b21ae83be8986c70b9f2e22dc3345e96ba6a

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CO	Code Optimization	Unresolved
	IFD	Incorrect Fee Deduction	Unresolved
	MEM	Misleading Error Messages	Unresolved
	MEM	Misleading Error Messages	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L16	Validate Variable Setters	Unresolved
	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Contract Name	Marvin
Testing Deploy	https://testnet.bscscan.com/address/0x56b3bce6290e70a351b3a966b8baa8fae81b1023
Symbol	MOB
Decimals	9
Total Supply	1,500,000,000,000
Badge Eligibility	Must Fix Criticals

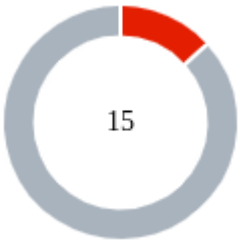
Audit Updates

Initial Audit	05 Jun 2024

Source Files

Filename	SHA256
marvin.sol	a016e00d69d68d32c37de47b0478b21ae83be8986c70b9f2e22dc3345e96ba6a

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

ST - Stops Transactions

Criticality	Critical
Location	src/marvin.sol
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.

For more information see PTRP.

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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	src/marvin.sol#L170
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setTxFees` function with a high percentage value.

```
function setTxFees(uint256 _marketingBuyFee, uint256
_marketingSellFee) public onlyOwner {
    marketingBuyFee = _marketingBuyFee;
    marketingSellFee = _marketingSellFee;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

CO - Code Optimization

Criticality	Minor / Informative
Location	src/marvin.sol#L134
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.

More specifically, at the constructor, the `uniswapV2Pair` is always address(0) which means the assignment of `uniswapV2Pair` to `_isExcludedFromLimits` list is redundant.

```
constructor(address __owner) Ownable(__owner) {
    ...
    _isExcludedFromLimits[uniswapV2Pair] = true;
    ...
}
```

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.

IFD - Incorrect Fee Deduction

Criticality	Minor / Informative
Location	src/marvin.sol#L204
Status	Unresolved

Description

The `_transfer` function of the contract is designed to apply a fee on token transfers. However, the current implementation may fail to correctly apply the sell fees in certain scenarios, specifically when `marketingBuyFee` is set to 0. The logic only checks if `marketingBuyFee` is greater than 0 before deducting any fees. This approach can result in the `marketingSellFee` not being applied if `marketingBuyFee` is zero, potentially leading to fee manipulation or incorrect fee deductions.

```
function _transfer(address sender, address recipient, uint256
amount) internal open(sender, recipient) {
    require(sender != address(0), "ERC20: transfer from the zero
address");
    require(recipient != address(0), "ERC20: transfer to the zero
address");
    require(_balances[sender] >= amount, "ERC20: Cannot send more
available balance");

    checkForMaxTxnLimit(sender, recipient, amount);

    if (!_isExcludedFromFee[sender] &&
!_isExcludedFromFee[recipient]) {
        if (marketingBuyFee > 0) {
            uint256 marketingTokens = (amount * marketingBuyFee) /
denominator;

            if (recipient == uniswapV2Pair) {
                marketingTokens = (amount * marketingSellFee) /
denominator;
            }

            _transferTokens(sender, address(this),
marketingTokens);
            amount = amount - marketingTokens;
        }
    }

    uint256 collectedFee = balanceOf(address(this));

    if (
        !inSwapAndLiquify && swapAndLiquifyEnabled && sender !=
address(uniswapV2Pair) && sender != address(this)
            && collectedFee > minimumTokensBeforeSwap &&
balanceOf(uniswapV2Pair) > 0
    ) {
        swapTokensForEth(minimumTokensBeforeSwap);
    }
    _transferTokens(sender, recipient, amount);
}
```

Recommendation

The team is advised to modify the logic to ensure that sell fees are always checked and applied independently of the buy fees. The fee calculation for sell transactions should not depend on the value of `marketingBuyFee`.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	src/marvin.sol#L254,258,278,284
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
function setExcludedFromLimit(address account, bool _enabled)
public onlyOwner
function setExcludedFromFee(address account, bool _enabled) public
onlyOwner
function includeToWhiteList(address[] memory _users) external
onlyOwner
function updateMarketingWalletAddress(address _marketingWallet)
external onlyOwner
```

Recommendation

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

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	src/marvin.sol#L177
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

In this case, the require statement is checking if the `maxTxLimit` is greater than 0.1% of the supply, but the error message is "To high Fee", while the error will be trhown when the fee is too low.

```
require(maxTxLimit > totalSupply / 1000, "To High Fee"); // 0.1%
```

Recommendation

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

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	src/marvin.sol#L160,230
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 _transfer(address sender, address recipient, uint256
amount) internal open(sender, recipient) {
    ...
    if (
        !inSwapAndLiquify && swapAndLiquifyEnabled && sender !=
address(uniswapV2Pair) && sender != address(this)
            && collectedFee > minimumTokensBeforeSwap &&
balanceOf(uniswapV2Pair) > 0
    ) {
        swapTokensForEth(minimumTokensBeforeSwap);
    }
    _transferTokens(sender, recipient, amount);
}
...
function swapTokensForEth(uint256 contractTokenBalance) public
lockTheSwap {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router),
contractTokenBalance);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        contractTokenBalance, 0, path, marketingWallet,
block.timestamp + 30
```

```
    );
}
```

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	src/marvin.sol#L160,284
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` 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 swapTokensForEth(uint256 contractTokenBalance) public
lockTheSwap {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router),
contractTokenBalance);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        contractTokenBalance, 0, path, marketingWallet,
block.timestamp + 30
    );
}
...
function updateMarketingWalletAddress(address _marketingWallet)
external onlyOwner {
    marketingWallet = payable(_marketingWallet);
}
```

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	src/marvin.sol#L160
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

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

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        contractTokenBalance, 0, path, marketingWallet,
block.timestamp + 30
    );
}
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	src/marvin.sol#L180,254,258
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 setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner
function setExcludedFromLimit(address account, bool _enabled)
public onlyOwner
function setExcludedFromFee(address account, bool _enabled) public
onlyOwner
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	src/marvin.sol#L89,90,91,92,110
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 totalSupply = 1500_000_000_000 * 10 ** 9
string public name = "Marvin"
string public symbol = "MOB"
uint256 public decimals = 9
uint256 public denominator = 100
```

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	src/marvin.sol#L70,170,175,180,254,258,278,284,288
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);
uint256 _marketingBuyFee
uint256 _marketingSellFee
uint256 _amount
bool _enabled
address[] memory _users
address _marketingWallet
```

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	src/marvin.sol#L171,172,176,289
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.

```
marketingBuyFee = _marketingBuyFee
marketingSellFee = _marketingSellFee;
maxTxLimit = _amount
minimumTokensBeforeSwap = _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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	src/marvin.sol#L285
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.

```
marketingWallet = payable(_marketingWallet)
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	src/marvin.sol#L292
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

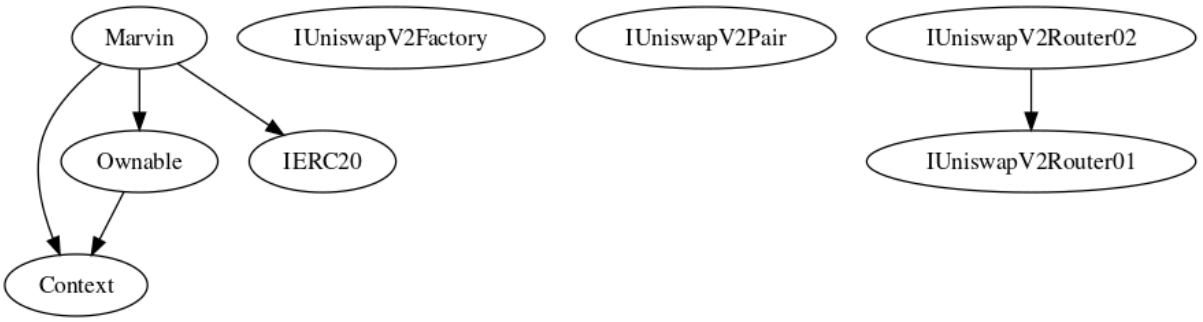
The team is advised to either remove the payable method or add a withdraw functionality. It is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

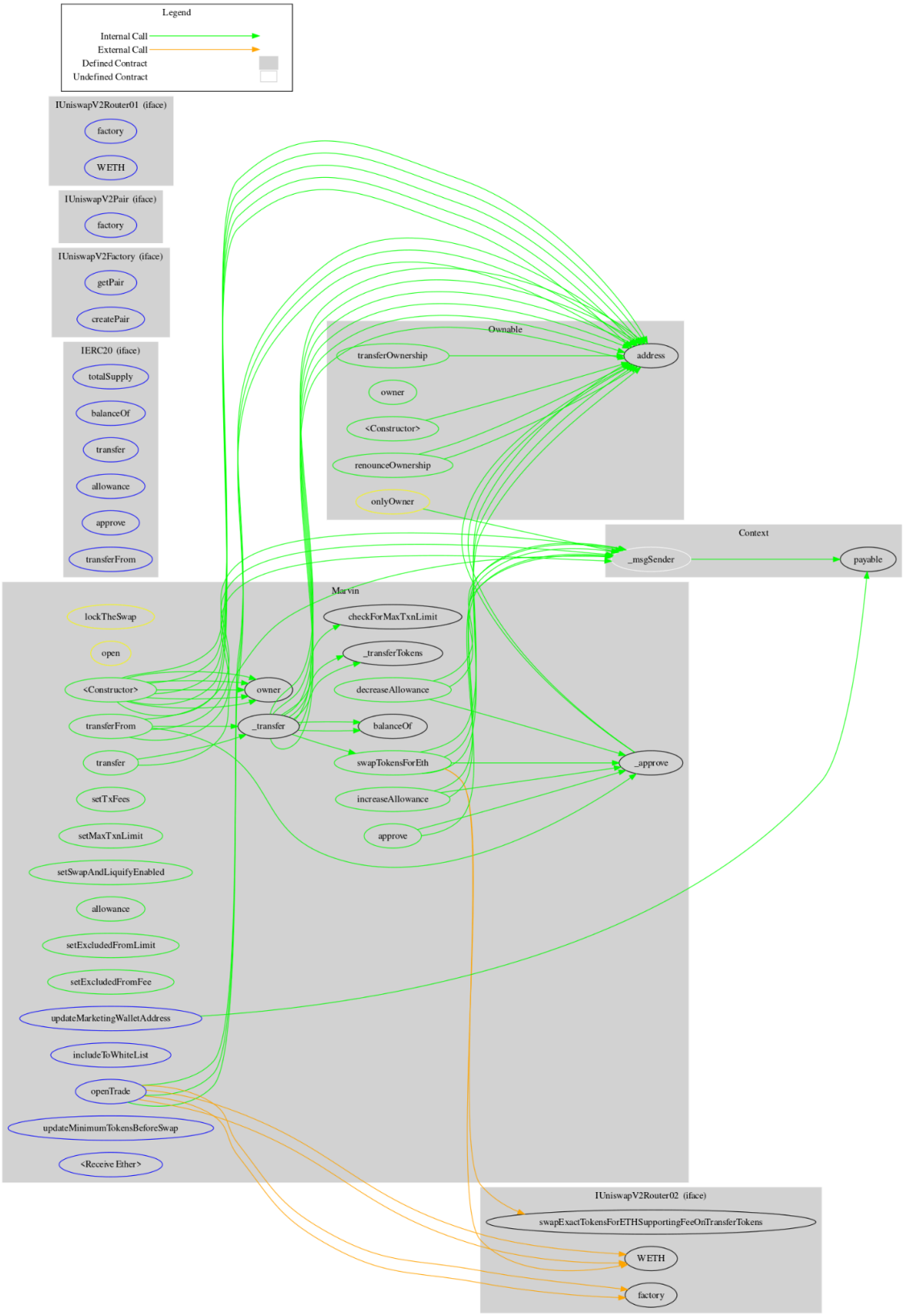
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Marvin	Implementation	Context, IERC20, Ownable		
		Public	✓	Ownable
	balanceOf	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	swapTokensForEth	Public	✓	lockTheSwap
	setTxFees	Public	✓	onlyOwner
	setMaxTxnLimit	Public	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	allowance	Public		-
	approve	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	open
	_transferTokens	Internal	✓	
	_approve	Internal	✓	
	checkForMaxTxnLimit	Internal		
	setExcludedFromLimit	Public	✓	onlyOwner

	setExcludedFromFee	Public	✓	onlyOwner
	openTrade	External	✓	onlyOwner
	includeToWhiteList	External	✓	onlyOwner
	updateMarketingWalletAddress	External	✓	onlyOwner
	updateMinimumTokensBeforeSwap	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01 (iface)
factory
WETH
IUniswapV2Pair (iface)
factory
IUniswapV2Factory (iface)
getPair
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
transferOwnership
address
owner
<Constructor>
renounceOwnership
onlyOwner
Context
_msgSender
payable
Marvin
lockTheSwap
open
<Constructor>
owner
transferFrom
_transfer
transfer
setTxFees
setMaxTxnLimit
setSwapAndLiquifyEnabled
allowance
setExcludedFromLimit
setExcludedFromFee
updateMarketingWalletAddress
includeToWhiteList
openTrade
updateMinimumTokensBeforeSwap
<Receive Ether>
checkForMaxTxnLimit
_transferTokens
decreaseAllowance
balanceOf
swapTokensForEth
_approve
increaseAllowance
approve
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory

Summary

Marvin Token is an interesting project that has a friendly and growing community. There are some functions that can be abused by the owner like manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. This audit investigates 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

https://www.cyberscope.io