Cyberscope

Audit Report

MOMO V2

August 2023

Network	ETH
Address	0x26d61aa110444668aa8dc6d601c83e0874ec1c2f
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ISD	Incorrect Sale Determination	Unresolved
Critical	ITMI	Invalid Transfer Method Implementation	Unresolved
Critical	TIO	Transfer Incorrect Operation	Unresolved
Minor / Informative	RS	Redundant Statements	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved
Minor / Informative	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Contract Name	MOMOV2
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Explorer	https://etherscan.io/address/0x26d61aa110444668aa8dc6d601c83e0874ec1c2f
Address	0x26d61aa110444668aa8dc6d601c83e0874ec1c2f
Network	ETH
Symbol	MOMO V2
Decimals	18
Total Supply	1,000,000,000,000

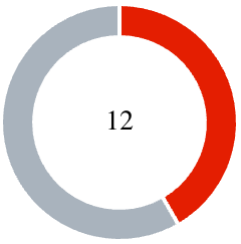
Audit Updates

Initial Audit	20 Aug 2023

Source Files

Filename	SHA256
MOMOV2.sol	2586fecd27a2573c3377a8f20cb08306caf0b3462efb7fb5c4aee7f2712eda79

Findings Breakdown



Critical	5
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	5	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

It has been identified that the contract exhibits critical code issues that could potentially lead to a complete halt of transfers. Two significant findings have highlighted a risk of infinite recursion in the overridden "transfer" method (see `ITMI`) and an incorrect operation of the "transferFrom" function (see `TIO`).

Recommendation

To address these critical issues and prevent a potential halt in transfers, it is essential to address the underlying problems. Rectifying the infinite recursion and correcting the "transferFrom" function's operation are paramount to restoring the contract's transfer functionality and ensuring the accurate and secure movement of tokens.

BC - Blacklists Addresses

Criticality	Critical
Location	MOMOV2.sol#L282
Status	Unresolved

Description

The contract employs a variable named "blacklist" to restrict certain addresses from performing transfers. However, the intended functionality aligns more closely with what is commonly referred to as a "whitelist." This inconsistency in variable naming introduces confusion and potentially leads to misunderstanding the contract's intended behavior.

```
require(isBlackListed[msg.sender], "User Blacklisted");
```

Recommendation

To address this finding and enhance code clarity and readability, it is strongly recommended to align the variable's naming with its intended functionality. In this case, renaming the variable from "blacklist" to "whitelist" would better reflect the contract's purpose and align with industry-standard terminology.

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

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

ISD - Incorrect Sale Determination

Criticality	Critical
Location	MOMOV2.sol#L281
Status	Unresolved

Description

The contract utilizes an inaccurate approach for comparing the recipient address with the router address to ascertain a sale. In typical scenarios, a sale is typically executed when the recipient address matches the pair address. However, the contract's flawed implementation of this comparison introduces the risk of inaccurate sale determination.

```
if (to == router) {
 ...
}
```

Recommendation

The team is adviced to implement the standard approach of comparing the recipient address with the pair address to identify sales. This aligns with industry norms and established practices, ensuring accurate recognition of sales transactions.

ITMI - Invalid Transfer Method Implementation

Criticality	Critical
Location	MOMOV2.sol#L281
Status	Unresolved

Description

The contract's implementation of the "transfer" method does not correctly call the parent transfer, leading to an infinite recursion loop when attempting to execute transfers. This flaw in the contract's logic results in a serious operational breakdown, causing transactions to become stuck in an endless cycle.

```
function _transfer(address from,address to,uint256 amount) internal
virtual override {
    ...
    if (to == router) {
        ...
        _transfer(owner, to, amount - taxAmount); // To wallet
address
    } else {
        super._transfer(from, to, amount);
    }

    _afterTokenTransfer(from, to, amount);
}
```

Recommendation

The team is adviced to correct the implementation of the overridden "transfer" method. The corrected override should include a proper call to the parent "transfer" method to ensure the execution of the original transfer functionality.

TIO - Transfer Incorrect Operation

Criticality	Critical
Location	MOMOV2.sol#L283
Status	Unresolved

Description

The contract currently employs the "msg.sender" address in the transfer function instead of utilizing the appropriate sender address. Consequently, the "transferFrom" function experiences incorrect behavior, as it inadvertently reduces the amount from the "msg.sender" instead of the intended sender, leading to erroneous financial transactions.

```
if (isWhiteListed[msg.sender]) {
    // no tax
    address owner = _msgSender();
    _transfer(owner, to, amount);
} else {
    // tax
    address owner = _msgSender();
    uint256 taxAmount = (amount * sellTaxRate) / 10000; //
Calculate tax
    _transfer(owner, taxWallet, taxAmount); // For tax
    _transfer(owner, to, amount - taxAmount); // To wallet address
}
```

Recommendation

To address this finding and rectify the incorrect operation, it is crucial to replace the usage of "msg.sender" with the appropriate sender address within the transfer function. The function's logic should align with the contract's intention, ensuring that the sender's balance is accurately adjusted during token transfers.

RS - Redundant Statements

Criticality	Minor / Informative
Location	MOMOV2.sol#L213
Status	Unresolved

Description

Redundant statements are statements that are unnecessary or have no effect on the contract's behavior. These can include declarations of variables or functions that are not used, or assignments to variables that are never used.

As a result, it can make the contract's code harder to read and maintain, and can also increase the contract's size and gas consumption, potentially making it more expensive to deploy and execute.

The variable `buyTaxRate` is not used from the contract.

```
uint256 public buyTaxRate;  // Tax rate for buys (in basis points)
```

Recommendation

To avoid redundant statements, it's important to carefully review the contract's code and remove any statements that are unnecessary or not used. This can help to improve the clarity and efficiency of the contract's code.

By removing unnecessary or redundant statements from the contract's code, the clarity and efficiency of the contract will be improved. Additionally, the size and gas consumption will be reduced.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MOMOV2.sol#L226,227
Status	Unresolved

Description

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

```
router
taxWallet
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MOMOV2.sol#L212,213,230,239,258,263
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.

```
address[] private WhiteListUsers
address[] private BlackListUsers
address[] memory _users
uint256 _taxRate
```

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	MOMOV2.sol#L265
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.

```
sellTaxRate = _taxRate
```

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	MOMOV2.sol#L226
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.

```
router = _router
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MOMOV2.sol#L2
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 ^0.8.9;
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	MOMOV2.sol#L256
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 {}
```

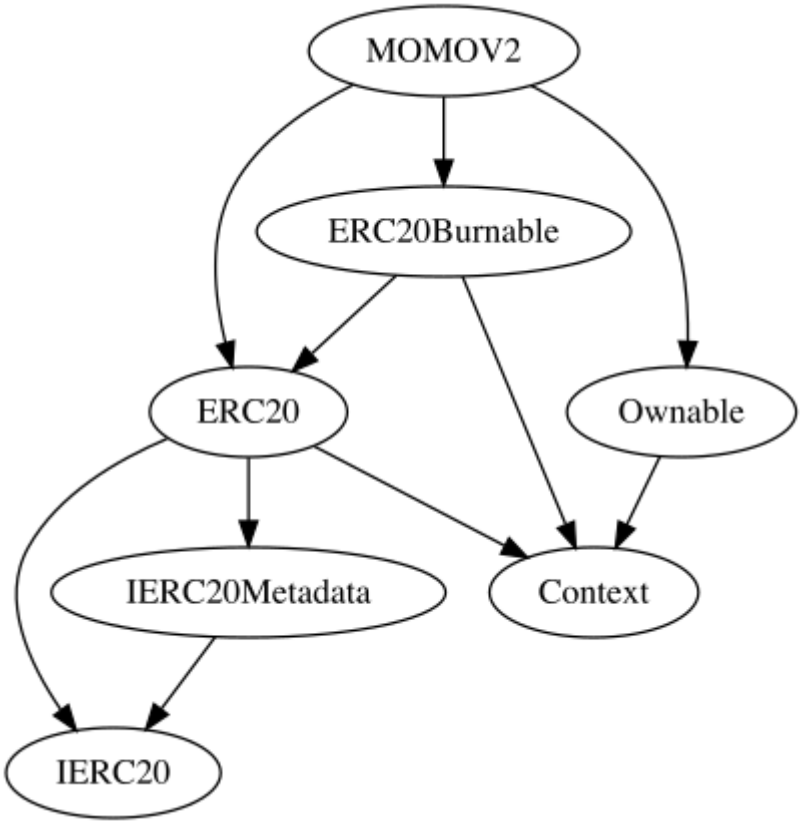
Recommendation

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.

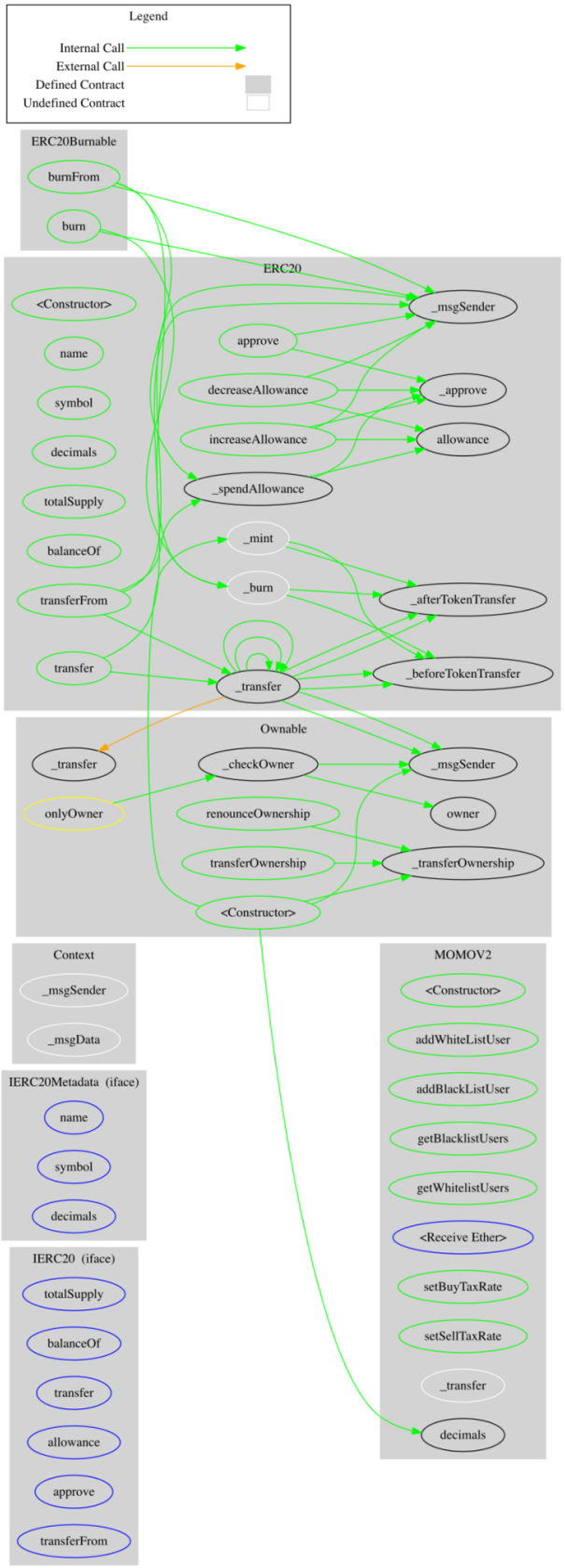
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MOMOV2	Implementation	ERC20, ERC20Burnable, Ownable		
		Public	✓	ERC20
	addWhiteListUser	Public	✓	onlyOwner
	addBlackListUser	Public	✓	onlyOwner
	getBlacklistUsers	Public		-
	getWhitelistUsers	Public		-
		External	Payable	-
	setBuyTaxRate	Public	✓	onlyOwner
	setSellTaxRate	Public	✓	onlyOwner
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20Burnable
burnFrom
burn
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
approve
decreaseAllowance
increaseAllowance
_spendAllowance
_mint
_burn
_transfer
_msgSender
_approve
allowance
_afterTokenTransfer
_beforeTokenTransfer
Ownable
_transfer
onlyOwner
_checkOwner
renounceOwnership
transferOwnership
<Constructor>
_msgSender
owner
_transferOwnership
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
MOMOV2
<Constructor>
addWhiteListUser
addBlackListUser
getBlacklistUsers
getWhitelistUsers
<Receive Ether>
setBuyTaxRate
setSellTaxRate
_transfer
decimals

Summary

MOMO V2 contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 10% 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