



Cyberscope

Audit Report

AlphaGate

October 2023

SHA256 e9708346fb64808e2a5a49d27a1692f08d46ed6a78dfe4d12a50d8c3b84ea741

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
●	RCS	Redundant Conditional Statement	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	MU	Modifiers Usage	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
ST - Stops Transactions	7
Description	7
Recommendation	7
RCS - Redundant Conditional Statement	8
Description	8
Recommendation	8
PAV - Pair Address Validation	9
Description	9
Recommendation	9
RSW - Redundant Storage Writes	10
Description	10
Recommendation	10
MEE - Missing Events Emission	11
Description	11
Recommendation	11
MSC - Missing Sanity Check	12
Description	12
Recommendation	12
MU - Modifiers Usage	13
Description	13
Recommendation	13
IDI - Immutable Declaration Improvement	14
Description	14
Recommendation	14
L02 - State Variables could be Declared Constant	15
Description	15
Recommendation	15
L04 - Conformance to Solidity Naming Conventions	16
Description	16
Recommendation	17
L07 - Missing Events Arithmetic	18
Description	18

Recommendation	18
L13 - Divide before Multiply Operation	19
Description	19
Recommendation	19
L16 - Validate Variable Setters	20
Description	20
Recommendation	20
L19 - Stable Compiler Version	21
Description	21
Recommendation	21
Functions Analysis	22
Inheritance Graph	26
Flow Graph	27
Summary	28
Disclaimer	29
About Cyberscope	30

Review

Contract Name	AlphaGate
Symbol	AGATE
Decimals	9
Total Supply	100,000,000

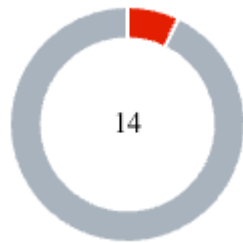
Audit Updates

Initial Audit	28 Oct 2023
---------------	-------------

Source Files

Filename	SHA256
AlphaGate.sol	e9708346fb64808e2a5a49d27a1692f08d46ed6a78dfe4d12a50d8c3b84ea741

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	AlphaGate.sol#L422
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 enabled the owner will not be able to disable them again.

```
if(!tradingOpen) {  
    require(isFeeExempt[sender] || isFeeExempt[recipient], "Trading is  
disabled");  
}
```

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.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	AlphaGate.sol#L451
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.

The `lowerLimitsAfter` and `lowerTaxesAfter` variables have both the same value and cannot be modified. Hence, checking both `totalTransactions < lowerLimitsAfter` and `totalTransactions < lowerTaxesAfter` conditions is redundant, as they will yield the same result.

```
if(totalTransactions < lowerLimitsAfter || totalTransactions <
lowerTaxesAfter) {
    totalTransactions++;
}
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	AlphaGate.sol#L290
Status	Unresolved

Description

The `setUniswapPair` function allows the contract owner to set the `uniswapPairAddress` to any arbitrary value without validation. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setUniswapPair(address pairAddress) external onlyOwner {  
    require(pairAddress != address(0), "ZeroAddress not allowed");  
    uniswapPairAddress = pairAddress;  
}
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	AlphaGate.sol#L287,300
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.

```
isFeeExempt[addressToExempt] = isExempt;  
tradingOpen = true;
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	AlphaGate.sol#L287,300
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
isFeeExempt[addressToExempt] = isExempt;  
tradingOpen = true;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	AlphaGate.sol#L282
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `finalMaxTxLimitPercent` must be greater than the `initialMaxTxLimitPercent`.

```
function setTxLimit(uint256 _newMaxTxLimitPercent) external onlyOwner {  
    finalMaxTxLimitPercent = _newMaxTxLimitPercent;  
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	AlphaGate.sol#L273,548
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(msg.sender == taxAddress, "Not TaxAddress");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	AlphaGate.sol#L258
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.

```
taxAddress
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AlphaGate.sol#L236,237,243,244,245,246
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.

```
string private _name = "AlphaGate"  
string private _symbol = "AGATE"  
uint256 private totalBuyTax = 4  
uint256 private totalSellTax = 4  
uint256 private totalBurnTax = 1  
uint256 private initialMaxTxLimitPercent = 1
```

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	AlphaGate.sol#L95,239,272,282,528,547
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 = 9  
uint256 _newSwapThresholdPercent  
uint256 _newMaxTxLimitPercent  
uint256 _tokenAmount
```

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	AlphaGate.sol#L274,283
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.

```
swapThresholdPercent = _newSwapThresholdPercent  
finalMaxTxLimitPercent = _newMaxTxLimitPercent
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	AlphaGate.sol#L430,447,449,500,503,505
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of precision.

```
maxTxTokens = (_totalSupply / 100) * finalMaxTxLimitPercent
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AlphaGate.sol#L258
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.

```
taxAddress = payable(_taxAddress)
```

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	AlphaGate.sol#L3
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 ^0.8.19;
```

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

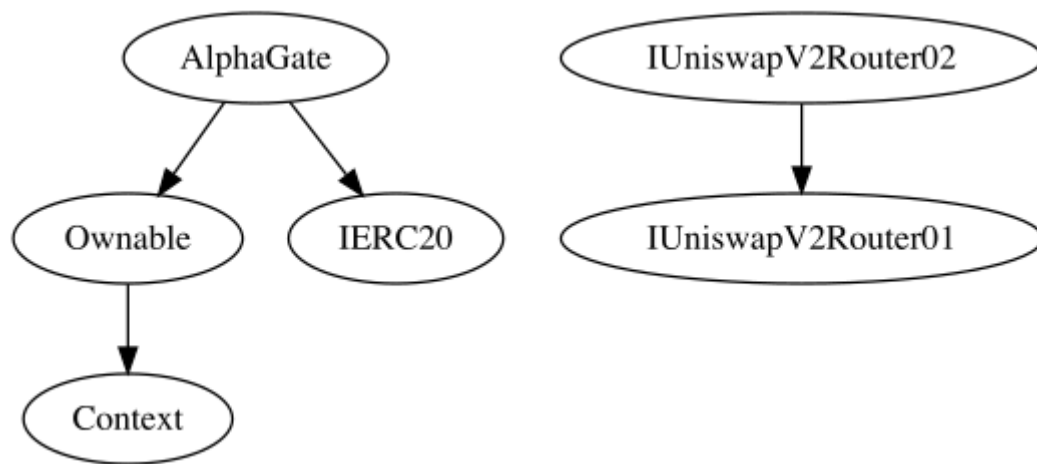
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner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

	transferFrom	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

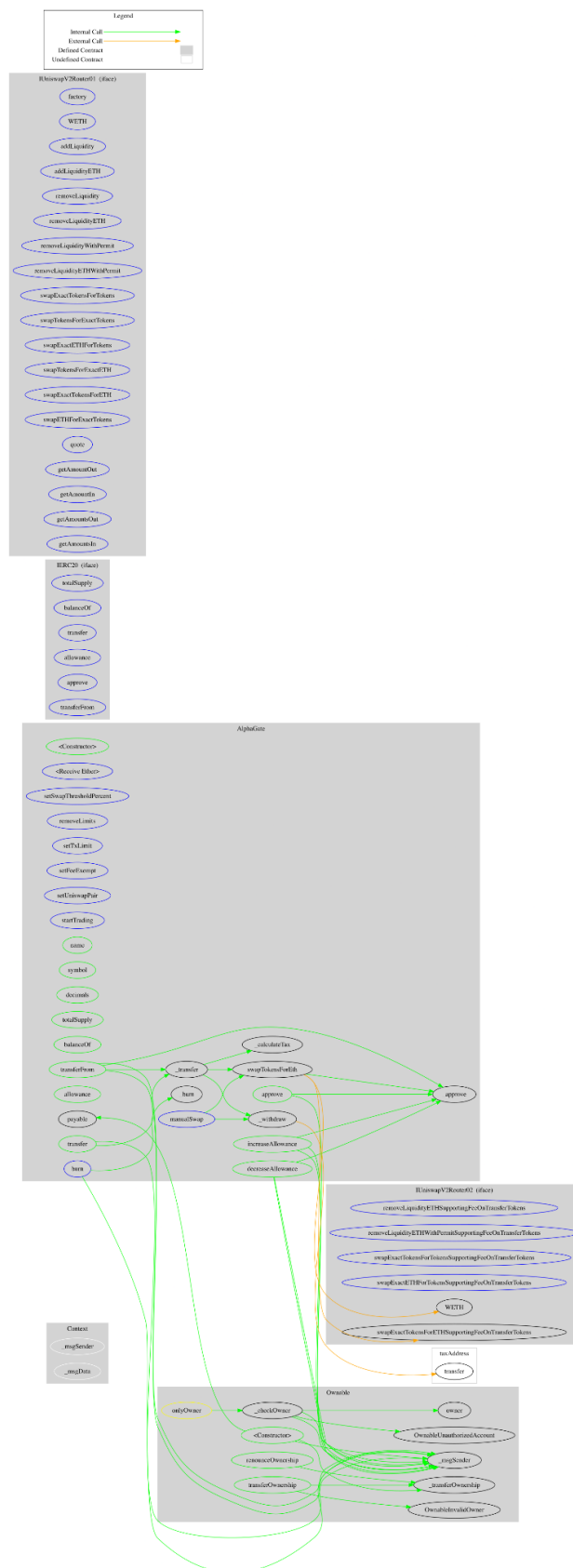
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
AlphaGate	Implementation	IERC20, Ownable		
		Public	✓	-
		External	Payable	-
	setSwapThresholdPercent	External	✓	-
	removeLimits	External	✓	onlyOwner
	setTxLimit	External	✓	onlyOwner
	setFeeExempt	External	✓	onlyOwner
	setUniswapPair	External	✓	onlyOwner
	startTrading	External	✓	onlyOwner
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_calculateTax	Internal		
	_burn	Internal	✓	
	burn	External	✓	-
	swapTokensForEth	Internal	✓	
	_withdraw	Internal	✓	
	manualSwap	External	✓	-
	_approve	Internal	✓	

Inheritance Graph



Flow Graph



Summary

AlphaGate 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. There is also a limit of max 5% fees buy and sell fees. Additionally, the contract implements a fee mechanism where that charges users with 25% buy fees and 35% sell fees for the first 20 transactions.

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>