



Cyberscope

Audit Report

KUSH ALIENZ

March 2023

Type	ERC20
Network	ARBITRUM
Address	0x57a1716703bc43f5e065085ea66f35b22a0990a3
Audited by	© cyberscope

Table of Contents

Table of Contents	1
Review	3
Analysis	4
BC - Blacklists Addresses	5
Description	5
Recommendation	5
Diagnostics	6
L01 - Public Function could be Declared External	7
Description	7
Recommendation	7
L02 - State Variables could be Declared Constant	8
Description	8
Recommendation	8
L04 - Conformance to Solidity Naming Conventions	9
Description	9
Recommendation	9
L07 - Missing Events Arithmetic	11
Description	11
Recommendation	11
L09 - Dead Code Elimination	12
Description	12
Recommendation	12
L13 - Divide before Multiply Operation	13
Description	13
Recommendation	13
L14 - Uninitialized Variables in Local Scope	14
Description	14
Recommendation	14
L19 - Stable Compiler Version	15
Description	15
Recommendation	15
L20 - Succeeded Transfer Check	16
Description	16
Recommendation	16
Functions Analysis	17
Inheritance Graph	21
Flow Graph	22

Summary	23
Disclaimer	24
About Cyberscope	25

Review

Contract Name	KushAlienz
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://arbiscan.io/address/0x57a1716703bc43f5e065085ea66f35b22a0990a3
Address	0x57a1716703bc43f5e065085ea66f35b22a0990a3
Network	ARBITRUM
Symbol	KALIEN
Decimals	18
Total Supply	1,000,000

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

BC - Blacklists Addresses

Criticality	Critical
Location	contract.sol#L902
Status	Unresolved

Description

The contract owner has the authority to massively stop addresses from transactions. The owner may take advantage of it by calling the `setBulkIsBlacklisted` function.

```
function setBulkIsBlacklisted( address[] calldata accounts, bool _state )
    external onlyOwner { for (uint256 i = 0; i < accounts.length; ) {
        isBlacklisted[accounts[i]] = _state; unchecked { i++; } } }
```

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 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.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	L01	Public Function could be Declared External	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
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

L01 - Public Function could be Declared External

Criticality	Minor / Informative
Location	contract.sol#L118,126,143,150,182,440,444,584,592,609,621,635,948
Status	Unresolved

Description

A public function is a function that can be called from external contracts or from within the contract itself. An external function is a function that can only be called from external contracts, and cannot be called from within the contract itself.

It's generally a good idea to declare functions as external if they are only intended to be called from external contracts, as this can help make the contract's code easier to understand and maintain. Declaring a function as external can also help to improve the contract's performance and gas consumption.

```
function name() public view virtual override returns (string memory) {
    return _name;
}

function symbol() public view virtual override returns (string memory) {
    return _symbol;
}
...
function decimals() public view virtual override returns (uint8) {
    return 18;
}

function totalSupply() public view virtual override returns (uint256) {
    return _totalSupply;
}
...
```

Recommendation

It's important to choose the appropriate visibility for each function based on how it is intended to be used. Declaring a function as external when it should be public, or vice versa can lead to unnecessary gas consumption.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contract.sol#L503,505,506
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.

```
address public routerAddress = 0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506
address public marketingAddress = 0x66b0B142D99cAF9d676A1511f7629fD4Bf215BCA
address public devAddress = 0x874c9d2C94662A679446236550294419891E4377
```

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	contract.sol#L92,94,466,501,507,744,835,840,849,861,874,875,883,891,897,904,914,918,924,934,943,957,958
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.

```
mapping(address => uint256) internal _balances
mapping(address => mapping(address => uint256)) internal _allowances
function WETH() external pure returns (address);
uint256 public genesis_block
address public UFOPoolAddress

...
```

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	contract.sol#L846,893,939,945
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.

```
tokenLiquidityThreshold = _newLiquidityThreshold * 1e18
coolDownTime = time * 1 seconds
maxBuyLimit = _maxBuy * 1e18
maxWalletLimit = _amount * 1e18
```

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	contract.sol#L348
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];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    _balances[account] = accountBalance - amount;
    _totalSupply -= amount;

    emit Transfer(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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contract.sol#L758,759,766,781,789,794
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 prediction.

```
uint256 devTokens = (swapTaxes.dev * tokenBalance) / feeswap
uint256 devAmt = (devTokens * afterBalance) / toSwap
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contract.sol#L693,696
Status	Unresolved

Description

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

```
uint256 feeswap  
Taxes memory currentTaxes
```

Recommendation

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contract.sol#L36
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 ^0.8.17;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contract.sol#L960
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_tokenAddress).transfer(_tokenAddress, _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

Functions Analysis

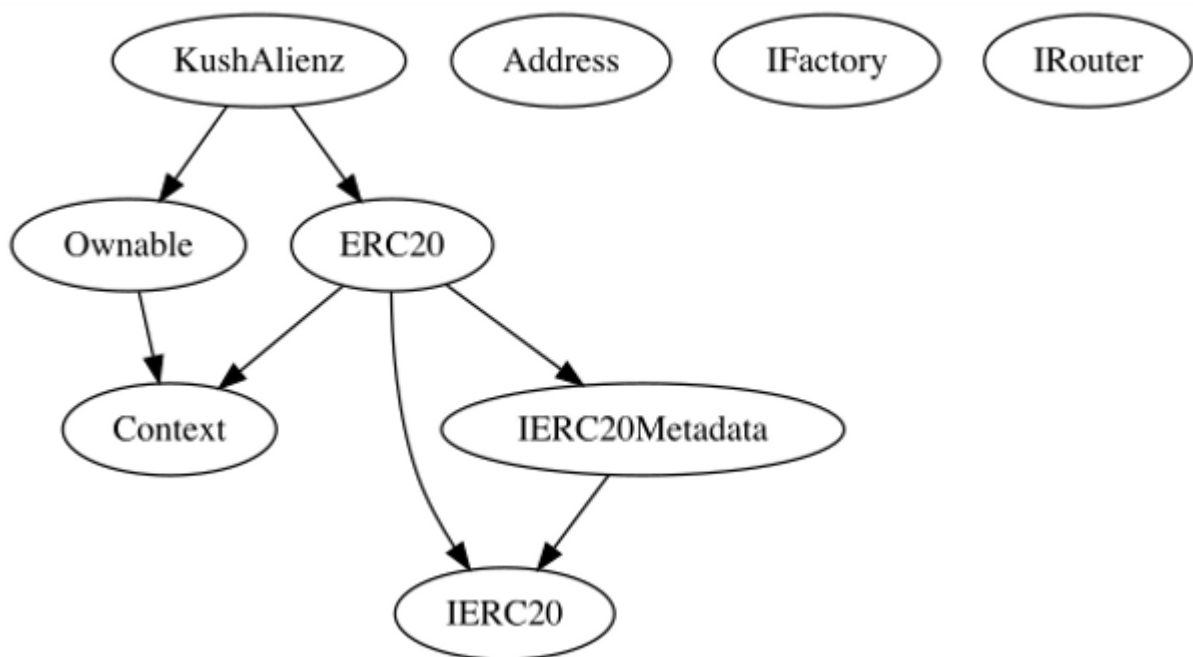
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
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	✓	
	_beforeTokenTransfer	Internal	✓	
Address	Library			
	sendValue	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IFactory	Interface			
	createPair	External	✓	-
IRouter	Interface			
	factory	External		-

	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
KushAlienz	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	startTrading	External	✓	onlyOwner
	approve	Public	✓	antiBot
	transferFrom	Public	✓	antiBot
	increaseAllowance	Public	✓	antiBot
	decreaseAllowance	Public	✓	antiBot
	transfer	Public	✓	antiBot
	_transfer	Internal	✓	
	handle_fees	Private	✓	mutexLock
	swapTokensForETH	Private	✓	
	addLiquidity	Private	✓	
	setLiquidityProvide	External	✓	onlyOwner
	setLiquidityThreshold	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setSellTaxes	External	✓	onlyOwner
	setRouterAndPair	External	✓	onlyOwner
	setUFOPoolAddress	External	✓	onlyOwner
	setCooldown	External	✓	onlyOwner
	setIsBlacklisted	External	✓	onlyOwner
	setBulkIsBlacklisted	External	✓	onlyOwner
	setAllowedTransfer	External	✓	onlyOwner
	setExemptFee	External	✓	onlyOwner
	setBulkExemptFee	External	✓	onlyOwner
	setMaxTxLimit	External	✓	onlyOwner

	setMaxWalletlimit	External	✓	onlyOwner
	getPair	Public		-
	rescueETH	External	✓	onlyOwner
	rescueBEP20	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

KUSH ALIENZ 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 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>