Cyberscope

Audit Report
KToken

January 2023

SHA256	62a79e90baedcd55a294ea3729180537be7bb6fdfb0c2f9016d7669f922869c4
Audited by	© cyberscope

Table of Contents

Review

Contract Name	KToken
Testing Deploy	https://testnet.bscscan.com/address/0x97a58f296b5e75d5907fbca9575f8296dd458be4
Symbol	TST
Decimals	9
Total Supply	0

Audit Updates

Initial Audit	19 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/security/Pausable.sol	2072248d2f79e661c149fd6a6593a8a3f038466557c9b75e50e0b001bcb5cf97
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef8c4e410ae99608be5964d98fa701
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	0344809a1044e11ece2401b4f7288f414ea41fa9d1dad24143c84b737c9fc02e
@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol	00d9364a71bfb7590fdeb7e097fe84159f4fc002c4f603b036c61f91e6368861
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
contracts/KToken.sol	62a79e90baedcd55a294ea3729180537be7bb6fdfb0c2f9016d7669f922869c4

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Status	Unresolved

Description

The contract is initialized with `sellPenaltyEnabled_` enabled. As a result, the fees are over the allowed limit of 25% to 33%. The owner can disable the sell penalty by calling the `disableSellPenalty` function.

```
bool public sellPenaltyEnabled_ = true;

function disableSellPenalty() public onlyOwner {
  require(
    sellPenaltyEnabled_ == true,
    "KToken: sell penalty is already disabled"
  );
  sellPenaltyEnabled_ = false;
}
```

Recommendation

We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ST - Stops Transactions

Criticality	Minor / Informative
Location	contracts/KToken.sol#L141
Status	Unresolved

Description

The contract owner has the authority to pause and unpause the transactions for all users. The owner may take advantage of it by calling the `pause` function.

```
function _beforeTokenTransfer(
  address from,
  address to,
  uint256 amount
) internal override whenNotPaused {
  super._beforeTokenTransfer(from, to, amount);
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

MT - Mints Tokens

Criticality	Critical
Location	contracts/KToken.sol#L98
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) public onlyOwner {
    _mint(to, amount);
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

🔴 Critical 🟡 Medium ● Minor / Informative

Severity	Code	Description	Status
●	BLC	Business Logic Concern	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L19	Stable Compiler Version	Unresolved

BLC - Business Logic Concern

Criticality	Minor / Informative
Location	contracts/KToken.sol#L62
Status	Unresolved

Description

The implementation may not follow the expected behavior. The function `setSellFee` changes the `buyFee_ = sellFee;` instead of the `sellFee_`.

```
function setSellFee(uint256 sellFee) public onlyOwner {
    require(
      sellFee > 0 && sellFee < 300,
      "KToken: buy fee must be between 0 and 300"
    );
    buyFee_ = sellFee;
}
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/KToken.sol#L17
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 sellFee_ = 300
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/KToken.sol#L59,67
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.

```
buyFee_ = buyFee
buyFee_ = sellFee
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/KToken.sol#L71,76,107
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(tradingEnabled_ == false, "KToken: trading is already enabled")

require(
      sellPenaltyEnabled_ == true,
      "KToken: sell penalty is already disabled"
    )
require(tradingEnabled_ == true, "KToken: trading is currently
disabled")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/KToken.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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Pausable	Implementation	Context		
		Public	✓	-
	paused	Public		-
	_requireNotPaused	Internal		
	_requirePaused	Internal		
	_pause	Internal	✓	whenNotPaused
	_unpause	Internal	✓	whenPaused
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-

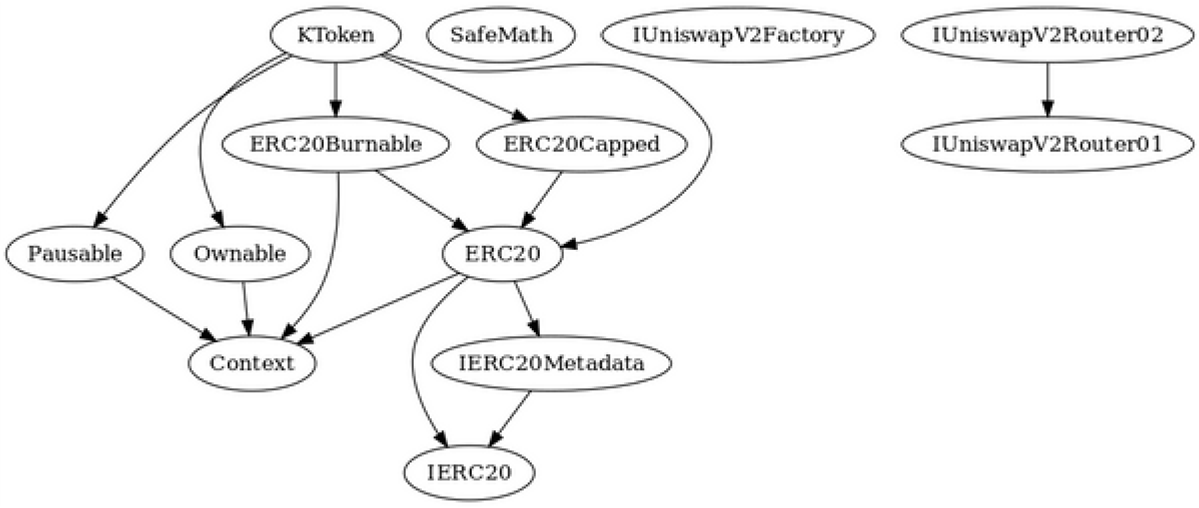
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
ERC20Capped	Implementation	ERC20		
		Public	✓	-
	cap	Public		-
	_mint	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IUniswapV2Factory	Interface			
	feeTo	External		-

	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	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	✓	-
KToken	Implementation	ERC20, ERC20Capped, ERC20Burnable, Pausable, Ownable		
		Public	✓	ERC20Capped ERC20
	decimals	Public		-
	setBuyFee	Public	✓	onlyOwner
	setSellFee	Public	✓	onlyOwner
	enableTrading	Public	✓	onlyOwner
	disableSellPenalty	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	includeInFees	Public	✓	onlyOwner
	setPancakePair	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_beforeTokenTransfer	Internal	✓	whenNotPaused

Inheritance Graph



Flow Graph

Summary

There are some functions that can be abused by the owner like stop transactions, manipulate the fees and mint tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. 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 2% for buy transaction fees and 3% for sell transaction.

Disclaimer

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