Cyberscope

Audit Report

Karpine Supply Chain xCellence

March 2024

Network	BSC
Address	0xc7cbf31bd4b3efd55b4cb85092152aca56e74fcc
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
●	RC	Redundant Calculations	Unresolved
●	RC	Redundant Check	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	KSCxToken
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://bscscan.com/address/0xc7cbf31bd4b3efd55b4cb85092152aca56e74fcc
Address	0xc7cbf31bd4b3efd55b4cb85092152aca56e74fcc
Network	BSC
Symbol	KSCx
Decimals	18
Total Supply	100,000,000,000
Badge Eligibility	Must Fix Criticals

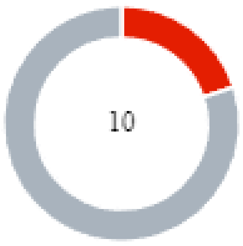
Audit Updates

Initial Audit	08 Mar 2024

Source Files

Filename	SHA256
KSCxToken.sol	2f711cca9852bcc85edcce893187ed7705dce5ed5632ccc4197e8fb4c9d097d7

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	8

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	KSCxToken.sol#L148,177
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users . The owner may take advantage of it by calling the `pause` function. As a result, the contract may operate as a honeypot.

```
function pause() external onlyOwner {
    _pause();
}

function _beforeTokenTransfer(address sender, address
recipient, uint256 amount) internal
override(ERC20,ERC20Pausable) whenNotPaused {
    super._beforeTokenTransfer(sender, recipient, amount);
}
```

Furthermore, the contract owner has the authority to stop the sales for all users by setting the `maxTransferAmountRate` or the `maxWalletBalanceRate` to a very low value.

```
uint256 maxTransferAmount = (totalSupply() *
maxTransferAmountRate * 10**decimals()) / (MAX_RATE *
10**decimals());
require(amount <= maxTransferAmount, "KSCxToken: Transfer
amount exceeds the maxTransferAmount");
uint256 recipientBalance = balanceOf(recipient);
uint256 maxWalletBalance = (totalSupply() *
maxWalletBalanceRate) / MAX_RATE;
require(recipientBalance + amount <= maxWalletBalance,

"KSCxToken: Exceeds maximum wallet token balance");
```

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.

BC - Blacklists Addresses

Criticality	Critical
Location	KSCxToken.sol#L69
Status	Unresolved

Description

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

```
function blacklist(address account, string memory reason)
external onlyOwner {
     require(!_blacklisted[account], "KSCxToken: Account is
already blacklisted");
     require(antiBotEnabled, "KSCxToken: Anti-bot feature is
disabled");
     _blacklisted[account] = true;
     _blacklistReasons[account] = reason;
     emit Blacklisted(account, reason);
}
```

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.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	KSCxToken.sol#L193,194
Status	Unresolved

Description

There are code segments in the contract, where the calculations are made, which multiply and divide by `10**decimals()`. These additional multiplications and divisions introduce unnecessary complexity and computational cost, without altering the outcome of the calculations.

```
uint256 feeAmount = (amount * transactionFee * 10**decimals()) / (100 *
10**decimals());
uint256 burnAmount = (amount * burnRate * 10**decimals()) / (100 *
10**decimals());
```

Recommendation

It is recommended to simplify the calculations by removing the redundant multiplication and division by `10**decimals()`. This simplification will reduce the gas cost associated with these operations, making the contract more efficient and will enhance the readability and maintainability of the contract code, because it removes unnecessary complexity.

RC - Redundant Check

Criticality	Minor / Informative
Location	KSCxToken.sol#L196
Status	Unresolved

Description

Within the `_transfer` function, a conditional statement is present to check that the `feeAmount`, `burnAmount`, and `amountAfterFeeAndBurn` exceed zero. This check is redundant, since there is no scenario where these conditions would not be satisfied.

```
uint256 feeAmount = (amount * transactionFee * 10**decimals()) / (100 *
10**decimals());
uint256 burnAmount = (amount * burnRate * 10**decimals()) / (100 *
10**decimals());
uint256 amountAfterFeeAndBurn = amount - feeAmount - burnAmount;
if (feeAmount > 0 && amountAfterFeeAndBurn> 0 && burnAmount > 0) {
```

Recommendation

It is recommended to re-evaluate the necessity of the conditional check within the `_transfer` function. Streamlining this conditional statement by removing unnecessary checks can enhance contract efficiency and readability, reducing gas costs for operations and minimizing potential confusion.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	KSCxToken.sol#L166
Status	Unresolved

Description

The `_transfer` function contains a `require` statement to prevent the transfer of tokens when the contract is paused. However, the contract also overrides the `_beforeTokenTransfer` function from `ERC20` and `ERC20Pausable` with an additional check for the paused state via the `whenNotPaused` modifier. The `whenNotPaused` modifier employs `_requireNotPaused()` to assert the contract is not paused. As a result the `require statement` in the `_transfer` function is redundant.

```
require(!paused(), "ERC20Pausable: token transfer while
paused");

function _beforeTokenTransfer(address sender, address
recipient, uint256 amount)
    internal override(ERC20,ERC20Pausable) whenNotPaused {
    super._beforeTokenTransfer(sender, recipient, amount);
}

modifier whenNotPaused() {
    _requireNotPaused();
    _;
}

function _requireNotPaused() internal view virtual {
    require(!paused(), "Pausable: paused");
}
```

Recommendation

It is recommended to remove the `require` statement from the `_transfer` function since the contract already implements the pause check mechanism.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	KSCxToken.sol#L64,92,107,113,125,131
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 updateMaxTransferAmountRate(uint256
_maxTransferAmountRate) external  onlyOwner {
    require(_maxTransferAmountRate <= MAX_RATE, "KSCxToken:
Transfer amount exceeds the maximum rate.");
    emit MaxTransferAmountRateUpdated(maxTransferAmountRate,
_maxTransferAmountRate);
    maxTransferAmountRate = _maxTransferAmountRate;
}

function updateMaxWalletBalanceRate(uint256
_maxWalletBalanceRate) external onlyOwner {
    require(_maxWalletBalanceRate <= MAX_RATE, "KSCxToken: Max
wallet balance exceeds the maximum rate.");
    emit MaxWalletBalanceRateUpdated(maxWalletBalanceRate,
_maxWalletBalanceRate);
    maxWalletBalanceRate = _maxWalletBalanceRate;
}
```

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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	KSCxToken.sol#L119
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `recoverERC20` function.

```
function recoverERC20(address tokenAddress, uint256
tokenAmount) external onlyOwner {
    IERC20(tokenAddress).transfer(owner(), tokenAmount);
    emit TokensRecovered(tokenAddress, tokenAmount);
}
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	KSCxToken.sol#L64,92,107,113,125,131,137
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.

```
bool _enabled
uint256 _maxTransferAmountRate
uint256 _maxWalletBalanceRate
uint256 _transactionFee
uint256 _burnRate
address _feeDestination
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	KSCxToken.sol#L15
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 ^0.8.20;
```

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	KSCxToken.sol#L120
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(owner(), tokenAmount)
```

Recommendation

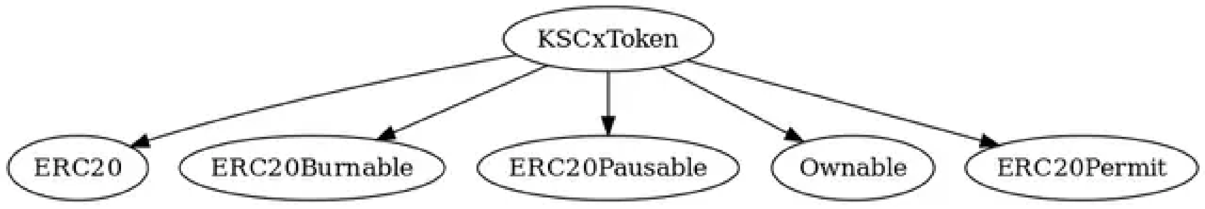
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

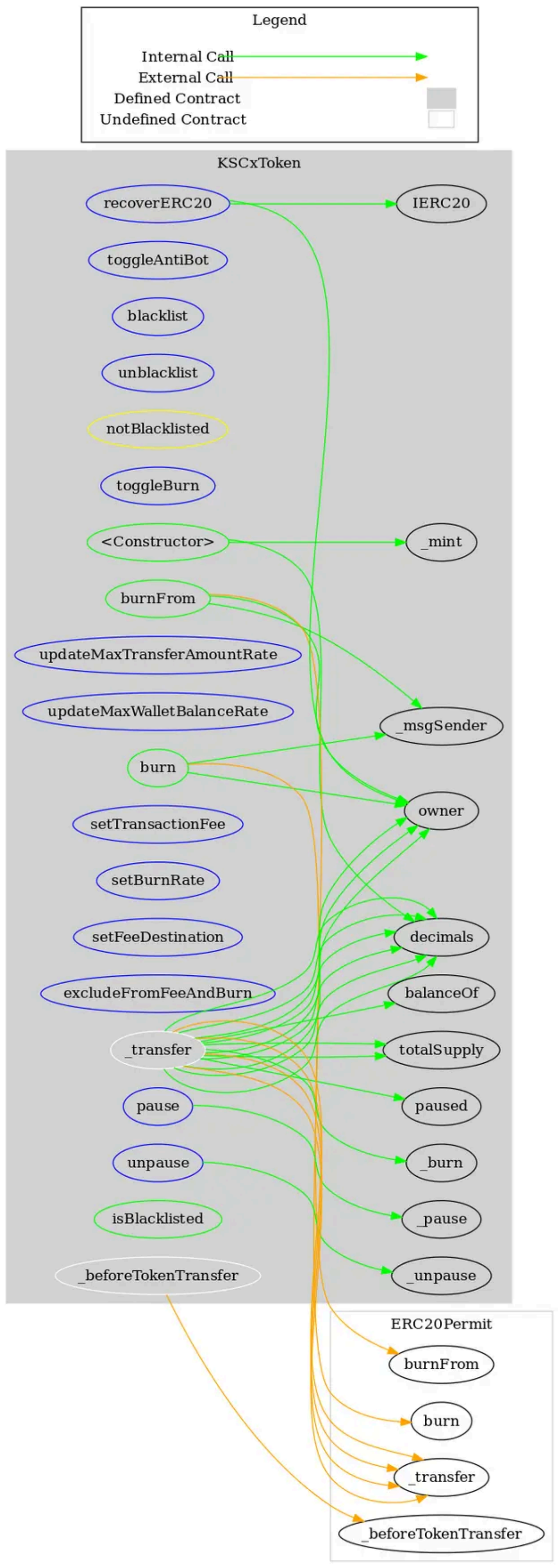
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
KSCxToken	Implementation	ERC20, ERC20Burnable, ERC20Pausable, Ownable, ERC20Permit		
		Public	✓	ERC20 Ownable ERC20Permit
	toggleAntiBot	External	✓	onlyOwner
	blacklist	External	✓	onlyOwner
	unblacklist	External	✓	onlyOwner
	toggleBurn	External	✓	onlyOwner
	burn	Public	✓	-
	burnFrom	Public	✓	-
	updateMaxTransferAmountRate	External	✓	onlyOwner
	updateMaxWalletBalanceRate	External	✓	onlyOwner
	recoverERC20	External	✓	onlyOwner
	setTransactionFee	External	✓	onlyOwner
	setBurnRate	External	✓	onlyOwner
	setFeeDestination	External	✓	onlyOwner
	excludeFromFeeAndBurn	External	✓	onlyOwner
	pause	External	✓	onlyOwner

	unpause	External	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	whenNotPaused
	isBlacklisted	Public		-
	_transfer	Internal	✓	notBlacklisted notBlacklisted

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
KSCxToken
recoverERC20
IERC20
toggleAntiBot
blacklist
unblacklist
notBlacklisted
toggleBurn
<Constructor>
_mint
burnFrom
updateMaxTransferAmountRate
updateMaxWalletBalanceRate
_msgSender
burn
owner
setTransactionFee
setBurnRate
setFeeDestination
decimals
excludeFromFeeAndBurn
balanceOf
_transfer
totalSupply
pause
paused
unpause
_burn
isBlacklisted
_pause
_beforeTokenTransfer
_unpause
ERC20Permit
burnFrom
burn
_transfer
_beforeTokenTransfer

Summary

Karpine Supply Chain xCellence 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.

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