Cyberscope

Audit Report

Crypto Scratch Cards

January 2024

Network	BSC
Address	0x447e6B899cF388B1df32C27a28F6f6ce6df90E07
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	Unresolved
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔵	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RFD	Redundant Function Declaration	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Crypto_Scratch_Cards
Compiler Version	v0.8.16+commit.07a7930e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x447e6b899cf388b1df32c27a28f6f6ce6df90e07
Address	0x447e6b899cf388b1df32c27a28f6f6ce6df90e07
Network	BSC
Symbol	CSC
Decimals	18
Total Supply	10,000,000,000
Badge Eligibility	Yes

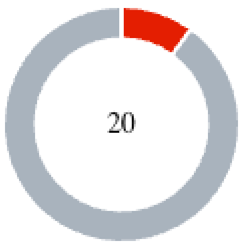
Audit Updates

Initial Audit	20 Jan 2024

Source Files

Filename	SHA256
Crypto_Scratch_Cards.sol	9a23fdebf154cbb3226b8371b606075cbc9eef1dbae3f1f46e9fe1cd81a5f0a6

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	18

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	18	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Crypto_Scratch_Cards.sol#L327,330,331
Status	Unresolved

Description

The contract owner has the authority to stop the transfers for all users excluding the authorized addresses. The owner may take advantage of it by setting the `transferFee` more than 100%.

```
uint256 feeAmount = amount.div(denominator).mul(getTotalFee(sender,
recipient));
...
return amount.sub(feeAmount);
```

The contract owner has the authority to stop the transactions for all users excluding the authorized addresses. The owner may take advantage of it by setting the `burnFee` to either 0 or a high value.

```
if(burnFee > uint256(0)){_transfer(address(this), address(DEAD),
amount.div(denominator).mul(burnFee));}
```

The contract owner has the authority to stop the sales for all users excluding the authorized addresses. The owner may take advantage of it by setting the `sellFee` to 1, `totalFee` to 0, and `burnFee` to `type(uint256).max`. Additionally, the issues, which are described in details, in sections PLPI and PTRP can also result in the suspension of sales. As a result, the contract may operate as a honeypot.

```
if(burnFee > uint256(0)){_transfer(address(this), address(DEAD),
amount.div(denominator).mul(burnFee));}
```

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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Crypto_Scratch_Cards.sol#L253,257
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setStructure` function with a high percentage value for the `burnFee` and `transferFee` variables.

```
function setStructure(uint256 _liquidity, uint256 _marketing, uint256
_burn, uint256 _development, uint256 _total, uint256 _sell, uint256
_trans) external onlyOwner {
    liquidityFee = _liquidity;
    marketingFee = _marketing;
    burnFee = _burn;
    developmentFee = _development;
    totalFee = _total;
    sellFee = _sell;
    transferFee = _trans;
    require(totalFee <= denominator.div(5) && sellFee <=
denominator.div(5), "totalFee and sellFee cannot be more than 20%");
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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	Crypto_Scratch_Cards.sol#L205,206
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[_address] = _enabled;
isFeeExempt[_address] = !_enabled;
```

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	Crypto_Scratch_Cards.sol#L205,206,219,224,229
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[_address] = _enabled;
isFeeExempt[_address] = !_enabled;
development_receiver = newwallet;
marketing_receiver = newwallet;
liquidity_receiver = newwallet;
```

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.

RFD - Redundant Function Declaration

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L206
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 contract contains a redundant function, `setisNoExempt` , that removes an address from the `isFeeExempt` mapping. This functionality is already implemented in the `setisExempt` function, which takes an `_enabled` argument to set the exemption state. Since the `_enabled` state is provided as an argument, the setisExempt function can be used for both setting and removing addresses, rendering the setisNoExempt function redundant.

```
function setisExempt(address _address, bool _enabled) external onlyOwner
{isFeeExempt[_address] = _enabled;}
function setisNoExempt(address _address, bool _enabled) external onlyOwner
{isFeeExempt[_address] = !_enabled;}
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L292
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForETH(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();
    _approve(address(this), address(router), tokenAmount);
    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp);
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L276,278
Status	Unresolved

Description

The contract sends funds to a `marketing_receiver` and `development_receiver` as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketing_receiver).transfer(marketingAmt);
payable(development_receiver).transfer(remainingBalance);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

MVN - Misleading Variables Naming

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

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

```
uint256 private totalFee = 1800;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L170
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 contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
bool private swapEnabled = true;
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L186
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.

```
pair
```

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	Crypto_Scratch_Cards.sol#L169,170
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 private denominator = 10000
bool private swapEnabled = true
```

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	Crypto_Scratch_Cards.sol#L94,147,151,152,153,178,179,180,205,206,210,250,281
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);

...
```

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	Crypto_Scratch_Cards.sol#L251
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.

```
liquidityFee = _liquidity
```

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	Crypto_Scratch_Cards.sol#L204
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 isCont(address addr) internal view returns (bool) {uint size;
assembly { size := extcodesize(addr) } return size > 0; }
```

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	Crypto_Scratch_Cards.sol#L272,273,275,327,330
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 feeAmount = amount.div(denominator).mul(getTotalFee(sender,
recipient))
```

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	Crypto_Scratch_Cards.sol#L78,219,224,229
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.

```
owner = adr
development_receiver = newwallet
marketing_receiver = newwallet
liquidity_receiver = newwallet
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L204
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(addr) }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Crypto_Scratch_Cards.sol#L23
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 ^0.8.16;
```

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	Crypto_Scratch_Cards.sol#L138
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, 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

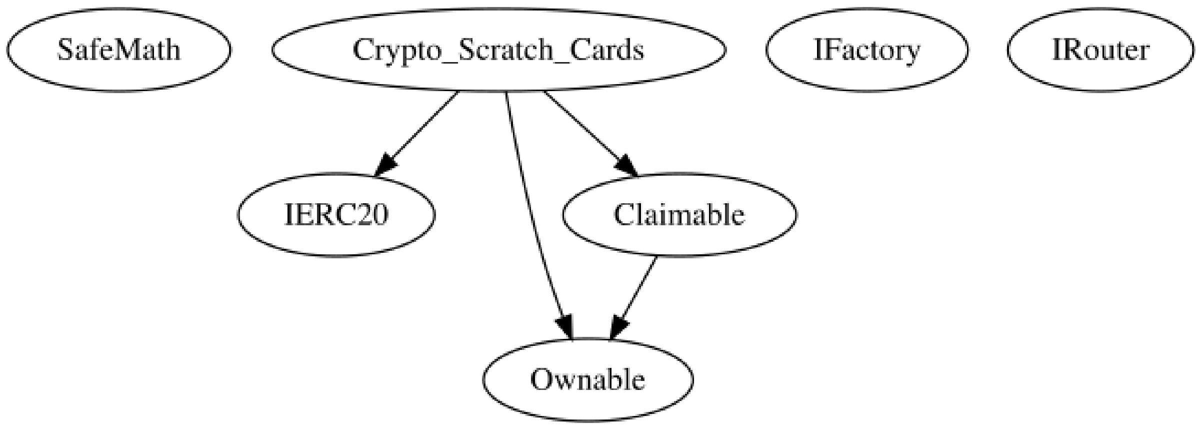
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-

	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation			
		Public	✓	-
	isOwner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-

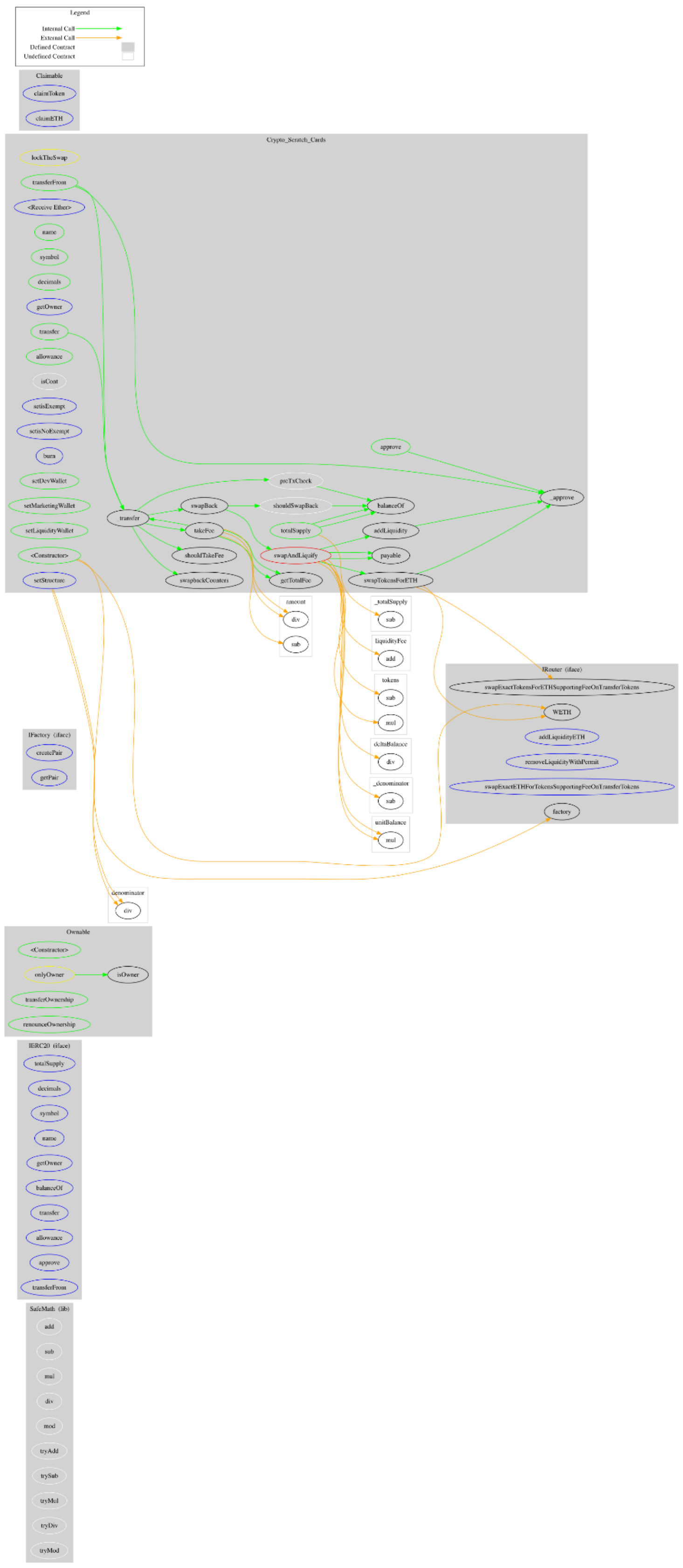
	removeLiquidityWithPermit	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Claimable	Implementation	Ownable		
	claimToken	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
Crypto_Scratch_Cards	Implementation	IERC20, Ownable, Claimable		
		Public	✓	Ownable
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	getOwner	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	isCont	Internal		
	setisExempt	External	✓	onlyOwner
	setisNoExempt	External	✓	onlyOwner
	approve	Public	✓	-

	totalSupply	Public		-
	burn	External	✓	-
	setDevWallet	Public	✓	onlyOwner
	setMarketingWallet	Public	✓	onlyOwner
	setLiquidityWallet	Public	✓	onlyOwner
	preTxCheck	Internal		
	_transfer	Private	✓	
	setStructure	External	✓	onlyOwner
	swapbackCounters	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	addLiquidity	Private	✓	
	swapTokensForETH	Private	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	
	shouldTakeFee	Internal		
	getTotalFee	Internal		
	takeFee	Internal	✓	
	transferFrom	Public	✓	-
	_approve	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Claimable
claimToken
claimETH
Crypto_Scratch_Cards
lockTheSwap
transferFrom
<Receive Ether>
name
symbol
decimals
getOwner
transfer
allowance
isCont
setisExempt
setisTxExempt
burn
setDevWallet
setMarketingWallet
setLiquidityWallet
<Constructor>
setStructure
approve
preTxCheck
swapBack
shouldSwapBack
balanceOf
takeFee
totalSupply
addLiquidity
shouldTakeFee
swapAndLiquify
payable
swapbackCounters
getTotalFee
swapTokensForETH
_approve
amount
div
sub
_totalSupply
liquidityFee
add
tokens
mul
deltaBalance
_denominator
initBalance
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
removeLiquidityWithPermit
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
IFactory (iface)
createPair
getPair
denominator
Ownable
onlyOwner
isOwner
transferOwnership
renounceOwnership
IERC20 (iface)
SafeMath (lib)
mod
tryAdd
trySub
tryMul
tryDiv
tryMod

Summary

Crypto Scratch Cards 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 stopping transactions and manipulating the fees. 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

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