Cyberscope

Audit Report

Crypto Delivery NFT Game

February 2024

Repository	https://github.com/cdeliverygamenft/CryptoDeliveryToken
Commit	bc09028c946b6225c13b2fbb63b250bae148b39f
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	IDI	Immutable Declaration Improvement	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	DCoin
Repository	https://github.com/cdeliverygamenft/CryptoDeliveryToken
Commit	bc09028c946b6225c13b2fbb63b250bae148b39f
Testing Deploy	https://testnet.bscscan.com/address/0xdede94dc8fae74d155f9be65ee446222b54df6a3
Symbol	DCOIN
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	13 Feb 2024

Source Files

Filename	SHA256
contracts/SwapHelper.sol	2a4c209206357aa4c08f968f91e8de016cad9725f14436d42d2457363c38e340
contracts/IPancake.sol	3da224a98892c96500bfdcc6841027ea931993aaec1b95cb634e3b5552d3107d
contracts/GasHelper.sol	f23354f144a002ae2d55f2f4ad06a6ac4ff6a1420c715c920fbfb221709ce78c

contracts/Events.sol	1e62a06260c92059257ddad9de7051ab30acc9a0c69d38a7a13001c60ff79be7
contracts/Errors.sol	bde04e5bceb6159b2242340ccd8571842dcbc8c35c705850cdb8d93914186e66
contracts/DCoin.sol	28504471b1589aae08f1e6b1fe2981703a7e2c65e8f12117747fe4f0ff5bc536
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/ERC20.sol	2d874da1c1478ed22a2d30dcf1a6ec0d09a13f897ca680d55fb49fbcc0e0c5b1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	2e6108a11184dd0caab3f3ef31bd15fed1bc7e4c781a55bc867ccedd8474565c
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Overview

Cyberscope audited a custom ERC-20 token from “The Crypto Delivery NFT Game”. The token consists of the following contracts: DCoin, GasHelper, Events, Errors, and SwapHelper. `DCoin` has additional features tailored for decentralized finance (DeFi) operations, specifically designed for integration with the PancakeSwap decentralized exchange (DEX) platform. The `SwapHelper` contract acts as an intermediary for handling fees collected from DCoin transactions. It allows for the withdrawal of accumulated WBNB to a specified fee receiver address. `GasHelper` provides utility functions for token transfer optimizations, swapping tokens through a liquidity pool, and querying token balances and liquidity pool reserves. These functions are designed to minimize gas costs through direct assembly code calls and efficient computation methods, crucial for DeFi transactions where efficiency and low transaction fees are highly valued.

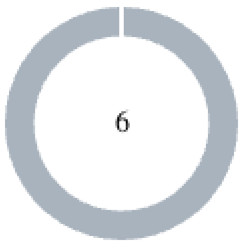
Overall, the `DCoin` smart contract and its associated helpers are designed to support an ecosystem where DCoin can be easily traded on DEXs, with built-in mechanisms for liquidity provision, fee collection, and gas optimization. These features are indicative of a sophisticated approach to creating a functional, efficient, and user-friendly DeFi token.

GasHelper Function Signatures

The `GasHelper` contract includes several functions that utilize assembly code calls to minimize gas costs. The function signature of each address utilized in the contract is shown below.

Address	Function Signature
0x0dfe1681	token0()
0xa9059cbb	transfer(address,uint256)
0x23b872dd	transferFrom(address,address,uint256)
0x022c0d9f	swap(uint256,uint256,address,bytes)
0x70a08231	balanceOf(address)
0x0902f1ac	getReserves()

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	6

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	6	0	0	0

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/SwapHelper.sol#L16
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.

```
feeReceiver
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/DCoin.sol#L52,57
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.

```
exceptFeeWallets[target] = status;
liquidityWallets[target] = status;
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/GasHelper.sol#L6
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.

```
uint internal constant swapFee = 25
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/GasHelper.sol#L31,44,74
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 tokenTransfer(address token, address recipient, uint amount) internal
{
    bool failed = false;
    assembly {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0xa9059cbb00000000000000000000000000000000000000000000000000000000)
      mstore(add(emptyPointer, 0x04), recipient)
      mstore(add(emptyPointer, 0x24), amount)
      failed := iszero(call(gas(), token, 0, emptyPointer, 0x44, 0, 0))
    }
    if (failed) revert("Unable to transfer token");
  }

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/GasHelper.sol#L20,33,46,60,76,89
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 {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0x0dfe168100000000000000000000000000000000000000000000000000000000)
      failed := iszero(staticcall(gas(), pair, emptyPointer, 0x04,
emptyPointer, 0x20))
      token0 := mload(emptyPointer)
    }

assembly {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0xa9059cbb00000000000000000000000000000000000000000000000000000000)
      mstore(add(emptyPointer, 0x04), recipient)
      mstore(add(emptyPointer, 0x24), amount)
      failed := iszero(call(gas(), token, 0, emptyPointer, 0x44, 0, 0))
    }

...
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/SwapHelper.sol#L22
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(WBNB).transfer(feeReceiver, balance)
```

Recommendation

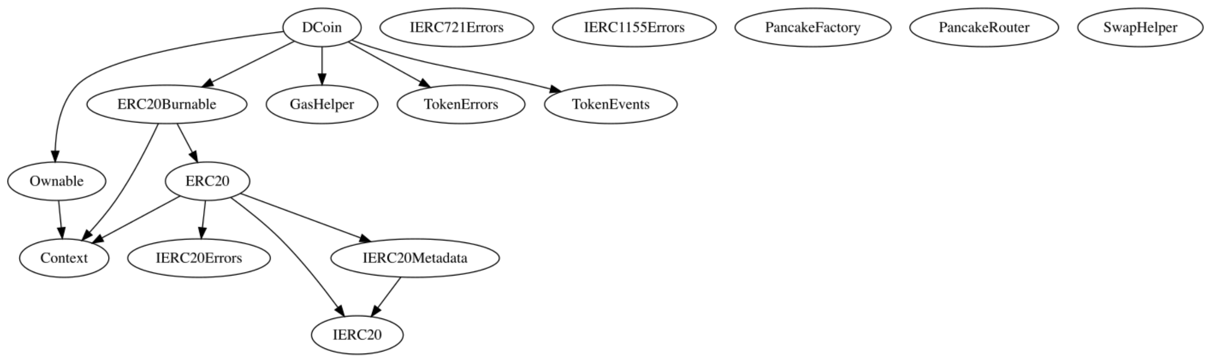
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

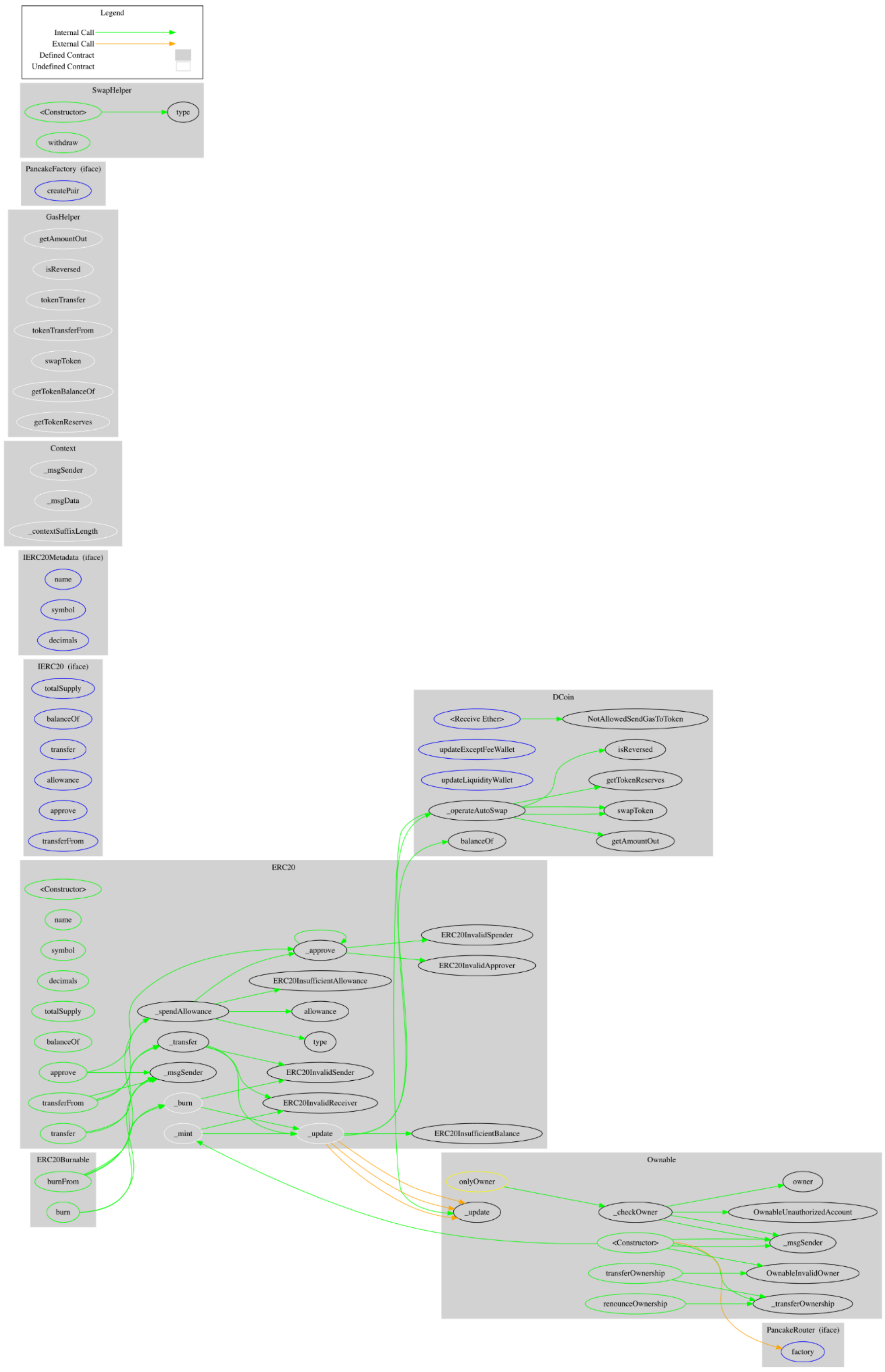
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SwapHelper	Implementation			
		Public	✓	-
	withdraw	Public	✓	-
GasHelper	Implementation			
	getAmountOut	Internal		
	isReversed	Internal		
	tokenTransfer	Internal	✓	
	tokenTransferFrom	Internal	✓	
	swapToken	Internal	✓	
	getTokenBalanceOf	Internal		
	getTokenReserves	Internal		
TokenEvents	Interface			
TokenErrors	Interface			

DCoin	Implementation	ERC20Burnable, GasHelper, TokenErrors, TokenEvents, Ownable		
		Public	✓	ERC20 Ownable
		External	Payable	-
	updateExceptFeeWallet	External	✓	onlyOwner
	updateLiquidityWallet	External	✓	onlyOwner
	_update	Internal	✓	
	_operateAutoSwap	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SwapHelper
<Constructor>
type
withdraw
PancakeFactory (iface)
createPair
GasHelper
getAmountOut
isReversed
tokenTransfer
tokenTransferFrom
swapToken
getTokenBalanceOf
getTokenReserves
Context
_msgSender
_msgData
_contextSuffixLength
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
DCoin
<Receive Ether>
NotAllowedSendGasToToken
updateExceptFeeWallet
isReversed
updateLiquidityWallet
getTokenReserves
_operateAutoSwap
swapToken
balanceOf
getAmountOut
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transferFrom
transfer
_approve
ERC20InvalidSpender
ERC20InvalidApprover
ERC20InsufficientAllowance
_spendAllowance
allowance
_transfer
type
_msgSender
ERC20InvalidSender
_burn
ERC20InvalidReceiver
_mint
_update
ERC20InsufficientBalance
ERC20Burnable
burnFrom
burn
Ownable
onlyOwner
owner
_update
_checkOwner
OwnableUnauthorizedAccount
<Constructor>
_msgSender
transferOwnership
OwnableInvalidOwner
renounceOwnership
_transferOwnership
PancakeRouter (iface)
factory

Summary

Crypto Delivery NFT Game contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Crypto Delivery NFT Game is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The fees are locked at 3% for both buys and sales.

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