Cyberscope

Audit Report

DigiToken

May 2024

Network	BSC
Address	0x0542b4C905F439a783b4B513B82a7e8c1537FfE1
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	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	AOI	Arithmetic Operations Inconsistency	Unresolved
	MEE	Missing Events Emission	Unresolved
	PAMAR	Pair Address Max Amount Restriction	Unresolved
	PMRM	Potential Mocked Router Manipulation	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Review

Contract Name	DigiFolioToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x0542b4c905f439a783b4b513b82a7e8c1537ffe1
Address	0x0542b4c905f439a783b4b513b82a7e8c1537ffe1
Network	BSC
Symbol	DGFL
Decimals	8
Total Supply	500,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	30 Apr 2024 https://github.com/cyberscope-io/audits/blob/main/dgfl/v1/audit.pdf
Corrected Phase 2	01 May 2024

Source Files

Filename	SHA256

DigiFolioToken.sol	49825456b0ca354dd78cf3933af3589c506222c24bfe02985c24a2985a3d0512

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	DigiFolioToken.sol#L296,310
Status	Unresolved

Description

Initially, the transactions are disabled for all users except the contract owner. The contract owner has to call the `enableTrading` function. Furthermore, the owner has the authority to stop the transactions for users by setting the `maxHoldLimit` to a very low value. Lastly, the owner has the authority to stop the transactions as described in the PAMAR finding. As a result, the contract may operate as a honeypot.

```
require(tradingActive, "Trading not started yet");

function enableTrading(address router, address pair) external
onlyOwner {
        tradingActive = true;
        uniswapRouter = IUniswapV2Router02(router);
        uniswapV2Pair = pair;
        _isExcludedFromLimit[router] = true;
        _isExcludedFromLimit[pair] = true;
    }

require(
  balanceOf(to) + amount  <= maxHoldLimit,
  "Max holding limit exceeds"
);

function setMaxHoldingLimit(uint256 _maxHoldLimit) external
onlyOwner {
        maxHoldLimit = _maxHoldLimit;
    }
```

Recommendation

The contract could embody a check for not allowing setting the `maxHoldLimit` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L301,317,318
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
_balances[from] = _balances[from].sub(amount);
_balances[to] = _balances[to].add(amount);

balanceOf(to) + amount <= maxHoldLimit
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L322,330,337
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.

```
function enableTrading(address router, address pair) external
onlyOwner {
    tradingActive = true;
    uniswapRouter = IUniswapV2Router02(router);
    uniswapV2Pair = pair;
    _isExcludedFromLimit[router] = true;
    _isExcludedFromLimit[pair] = true;
}

function setMaxHoldingLimit(uint256 _maxHoldLimit) external
onlyOwner {
    maxHoldLimit = _maxHoldLimit;
}
```

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.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L300,310
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
require(
    balanceOf(to) + amount  <= maxHoldLimit,
    "Max holding limit exceeds"
);
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L322
Status	Unresolved

Description

The contract does not have a check to ensure that the `enableTrading` function can be called only once. As a result, this function allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function enableTrading(address router, address pair) external
onlyOwner {
    tradingActive = true;
    uniswapRouter = IUniswapV2Router02(router);
    uniswapV2Pair = pair;
    _isExcludedFromLimit[router] = true;
    _isExcludedFromLimit[pair] = true;
}
```

Recommendation

It is recommended a check to be enforced, that allows the `openTrading` function to be called only once. Furthermore, 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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L45
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DigiFolioToken.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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L322,330,337
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 includeOrExcludeFromLimit(address _addr, bool _state)
    external
    onlyOwner
{
    _isExcludedFromLimit[_addr] = _state;
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L188
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 _tTotal = 500_000_000 * 10**_decimals
```

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	DigiFolioToken.sol#L176,183,185,187,192,193,326,333
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);
mapping(address => bool) public _isExcludedFromLimit
uint256 public _buyCount = 0
uint8 private constant _decimals = 8
string private constant _name = "DigiToken"
string private constant _symbol = "DGFL"
address _addr
bool _state
uint256 _maxHoldLimit
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	DigiFolioToken.sol#L200
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

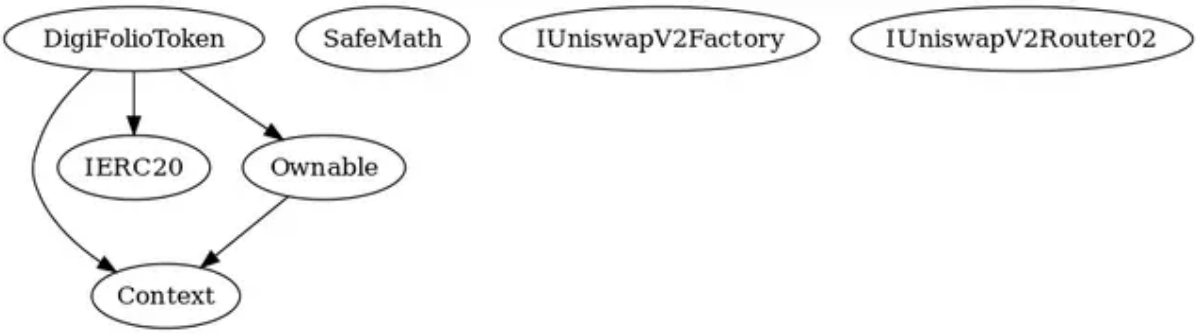
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

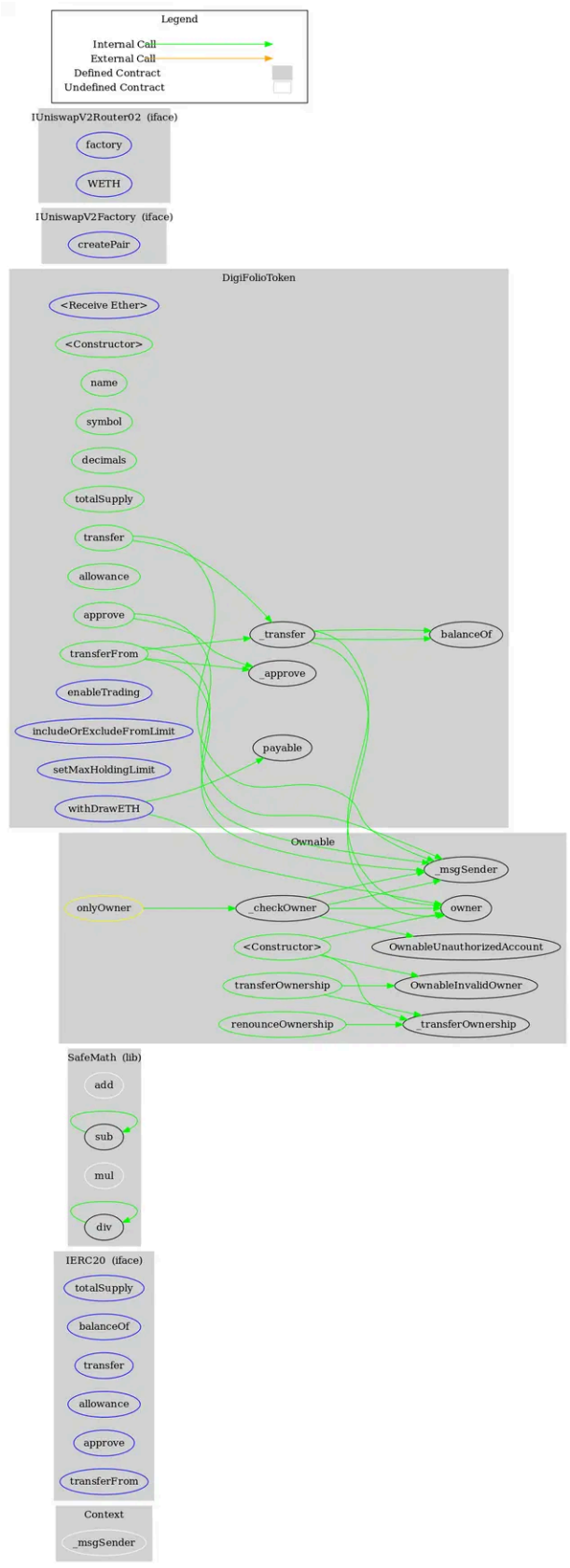
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	factory	External		-
	WETH	External		-
DigiFolioToken	Implementation	Context, IERC20, Ownable		
		External	Payable	-
		Public	✓	Ownable
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-

	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	enableTrading	External	✓	onlyOwner
	includeOrExcludeFromLimit	External	✓	onlyOwner
	setMaxHoldingLimit	External	✓	onlyOwner
	withDrawETH	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router02 (iface)
factory
WETH
IUniswapV2Factory (iface)
createPair
DigiFolioToken
<Receive Ether>
<Constructor>
name
symbol
decimals
totalSupply
transfer
allowance
approve
transferFrom
enableTrading
includeOrExcludeFromLimit
setMaxHoldingLimit
withDrawETH
_transfer
_approve
payable
balanceOf
Ownable
_msgSender
onlyOwner
_checkOwner
owner
<Constructor>
OwnableUnauthorizedAccount
transferOwnership
OwnableInvalidOwner
renounceOwnership
transferOwnership
SafeMath (lib)
add
sub
mul
div
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

DigiToken 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. 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