Cyberscope

Audit Report

Digits DAO

December 2022

SHA256	02e4ca7f027591040408ebd02e3138c82152fbcb01062ad27525fc410b1b4740
	5c0a11e7034476d2b611bc7e3031e8b0d8a4505fff481696afa8a754dac7d849
	d8f2bd09a76862457910aa7c6aa4152974e098d2411d93cb86fea24739724d0e
Audited by	© cyberscope

Table of Contents

Contract Review

Contract Names	Digits DividendTracker TokenStorage
Compiler Version	v0.8.10
Optimization	200 runs
Testing Deploy	Digits: https://testnet.bscscan.com/address/0xb304A7bDa5E583A0A10ec77A86072B00B7fF6121 DividendTracker: https://testnet.bscscan.com/address/0x52C2e0509444fdCacF2cc200C846aCf8481de2D6 TokenStorage: https://testnet.bscscan.com/address/0xA988b44d4Cd58dC03800f089fDBa7C9D24218e5e
Address	**Digits:** 0xb304A7bDa5E583A0A10ec77A86072B00B7fF6121 **DividendTracker:** 0x52C2e0509444fdCacF2cc200C846aCf8481de2D6 **TokenStorage:** 0xA988b44d4Cd58dC03800f089fDBa7C9D24218e5e
Network	BSC_TESTNET
Symbol	**Digits:** DIGITS **DividendTracker:** Digits_DividendTracker
Decimals	**Digits:** 18 **DividendTracker:** 18
Total Supply	**Digits:** 1000000000 **DividendTracker:** 0

Audit Updates

Initial Audit	19 Dec 2022

Source Files

Filename	SHA256
Digits.sol	02e4ca7f027591040408ebd02e3138c82 152fbcb01062ad27525fc410b1b4740
DividendTracker.sol	5c0a11e7034476d2b611bc7e3031e8b0d 8a4505fff481696afa8a754dac7d849
TokenStorage.sol	d8f2bd09a76862457910aa7c6aa415297 4e098d2411d93cb86fea24739724d0e

Introduction

Digits DAO consists of four contracts:

- Digits
- DividendTracker
- TokenStorage
- MultiRewards

This audit report is referring to the first three contracts. The audit report for the MultiRewards contract can be found at https://github.com/cyberscope-io/audits/tree/main/digits-dao/MultiRewards.pdf.

Roles

Digits

The Digits contract has two roles, the USER role and the OWNER role.

The OWNER role has the authority to

- Include/Exclude an address from fees.
- Set fees up to 24% combined.
- Enable/Disable fees.
- Claim all the balance of the contract.
- Renounce/Transfer ownership.

The USER role has the authority to

- Make transactions.
- Claim dividends distributed as DAI tokens.

DividendTracker

The DividendTracker contract has two roles, the USER role and the OWNER role.

The OWNER role has the authority to

- Renounce/Transfer ownership.
- Set the balance of an account.
- Include/Exclude an account from dividends.
- Transfer an accounts' dividends to that account.

The USER role has the authority to

- Transfer dividends to the contract.

TokenStorage

The DividendTracker contract has two roles, the MANAGER role and the OWNER role.

The OWNER role has the authority to

- Renounce/Transfer ownership.
- Add/Remove managers.

The MANAGER role has the authority to

- Transfer DAI to an address.
- Swap tokens for DAI.
- Distribute dividends.

Contract Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	RCC	Redundant Condition Check	Unresolved
●	CO	Code Optimization	Unresolved
●	US	Untrusted Source	Unresolved
🟠	ZD	Zero Division	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RCC - Redundant Condition Check

Criticality	Minor / Informative
Location	Digits.sol#L386
Status	Unresolved

Description

The variable `tokens` is an unsigned integer, which means its value is equal or greater than zero. Checking if it's lower or equal than zero is redundant.

```
if (tokens <= 0) {
  return;
}
```

Recommendation

The team is advised to change this code segment to check if the given value is only equal to zero.

CO - Code Optimization

Criticality	Minor / Informative
Location	Digits.sol#L262,263
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 variables `_maxTxAmount` and `_maxWallet` can be calculated on every transfer. Instead, they can be calculated once globally.

```
uint256 _maxTxAmount = (totalSupply() * maxTxBPS) / 10000;
uint256 _maxWallet = (totalSupply() * maxWalletBPS) / 10000;
```

Recommendation

The team is advised to take into consideration these segments 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.

US - Untrusted Source

Criticality	Minor / Informative
Location	Digits.sol#L117,440
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result it may produce security issues and harm the transactions.

```
dividendTracker = new DividendTracker(dai, address(this), uniswapRouter);
...
function setTokenStorage(address _tokenStorage) external onlyOwner {
  ...
  tokenStorage = ITokenStorage(_tokenStorage);
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

ZD - Zero Division

Criticality	Medium
Location	Digits.sol#L392
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert. The variable `totalFeeBPS` can be set to zero.

```
swapTokensMarketing = (tokens * treasuryFeeBPS) / totalFeeBPS;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of your contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero or should not allow executing of the corresponding statements.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Digits.sol#L602
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH. It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly in the triggered point, potentially leading to significant price volatility for the parties involved.

```
function updateDividendSettings(bool _swapEnabled, uint256 _swapTokensAtAmount, bool
_swapAllToken) external onlyOwner {
  swapEnabled = _swapEnabled;
  swapTokensAtAmount = _swapTokensAtAmount;
  swapAllToken = _swapAllToken;

  emit UpdateDividendSettings(_swapEnabled, _swapTokensAtAmount, _swapAllToken);
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	Digits.sol#L662
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 `rescueToken` function.

```
function rescueToken(address _token, uint256 _amount) external onlyOwner {
  IERC20(_token).transfer(msg.sender, _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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TokenStorage.sol#L38,43,110 DividendTracker.sol#L12,13,18 Digits.sol#L30,31,379,440,482,502,503,504,586,591,596,603,604,605,662,666
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 your Solidity code, making it easier for others to understand and work with.

The followings are 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 your code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _address
address _address
address _liquidityWallet
string private constant _name = "Digits_DividendTracker"
string private constant _symbol = "Digits_DividendTracker"
uint256 private constant magnitude = 2**128
string private constant _name = "Digits"
string private constant _symbol = "DIGITS"
address[] memory _users
address _tokenStorage
address _marketingWallet

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

You can find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	TokenStorage.sol#L48,56,75,98,111 Digits.sol#L534,541
Status	Unresolved

Description

The 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(
            managers[msg.sender] == true,
            "This address is not allowed to interact with the contract"
        )

require(
            managers[msg.sender] == true,
            "This address is not allowed to interact with the contract"
        )

require(
            managers[msg.sender] == true,
            "This address is not allowed to interact with the contract"
        )

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	DividendTracker.sol#L209,283 Digits.sol#L317,390,395
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in your contract. It's important to always initialize local variables with appropriate values before using them.

```
bool success
AccountInfo memory info
bool takeFee
uint256 swapTokensMarketing
uint256 swapTokensDividends
```

Recommendation

By initializing local variables before using them, you can help ensure that your contract functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	TokenStorage.sol#L31,32,33 DividendTracker.sol#L50,52 Digits.sol#L106,107,108
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.

```
dai = _dai
tokenAddress = _tokenAddress
liquidityWallet = _liquidityWallet
dai = _dai
tokenAddress = _tokenAddress
dai = _dai
uniswapRouter = _uniswapRouter
marketingWallet = _marketingWallet
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	TokenStorage.sol#L3 DividendTracker.sol#L3 Digits.sol#L3,4
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.10;
pragma solidity ^0.8.10;

pragma solidity ^0.8.10;
pragma experimental ABIEncoderV2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in. By including all required compiler options and flags in a single pragma directive, you can avoid conflicts and ensure that the contract can be compiled correctly.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	TokenStorage.sol#L52 DividendTracker.sol#L59,174 Digits.sol#L663
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(dai).transfer(to, amount)
IERC20(dai).transferFrom(msg.sender, address(this), daiDividends)
IERC20(dai).transfer(account, _withdrawableDividend)
IERC20(_token).transfer(msg.sender, _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**
ITokenStorage	Interface			
	swapTokensForDai	External	✓	-
	transferDai	External	✓	-
	addLiquidity	External	✓	-
	distributeDividends	External	✓	-
	setLiquidityWallet	External	✓	-
Digits	Implementation	Ownable, IERC20		
		Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	External		-
	approve	External	✓	-
	increaseAllowance	External	✓	-

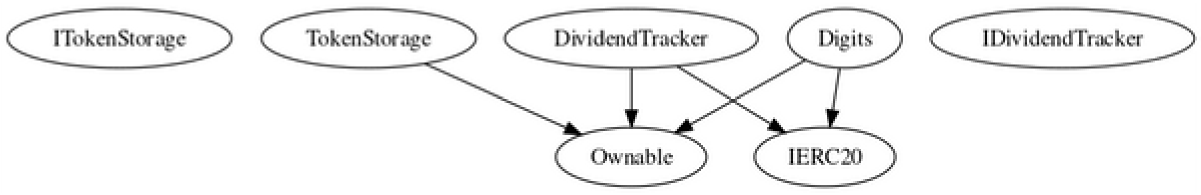
	decreaseAllowance	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transfer	Internal	✓	
	_executeTransfer	Private	✓	
	_approve	Private	✓	
	_mint	Private	✓	
	includeToWhiteList	Private	✓	
	_executeSwap	Private	✓	
	openTrading	External	✓	onlyOwner
	setTokenStorage	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	isExcludedFromFees	External		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	External		-
	setWallet	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	setFee	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	claim	External	✓	-
	compound	External	✓	-
	withdrawableDividendOf	External		-

	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
	getAccountInfo	External		-
	getLastClaimTime	External		-
	setSwapEnabled	External	✓	onlyOwner
	setTaxEnabled	External	✓	onlyOwner
	setCompoundingEnabled	External	✓	onlyOwner
	updateDividendSettings	External	✓	onlyOwner
	setMaxTxBPS	External	✓	onlyOwner
	excludeFromMaxTx	Public	✓	onlyOwner
	isExcludedFromMaxTx	External		-
	setMaxWalletBPS	External	✓	onlyOwner
	excludeFromMaxWallet	Public	✓	onlyOwner
	isExcludedFromMaxWallet	External		-
	rescueToken	External	✓	onlyOwner
	rescueETH	External	✓	onlyOwner
DividendTracker	Implementation	Ownable, IERC20		
		Public	✓	-
	distributeDividends	External	✓	-
	setBalance	External	✓	onlyOwner

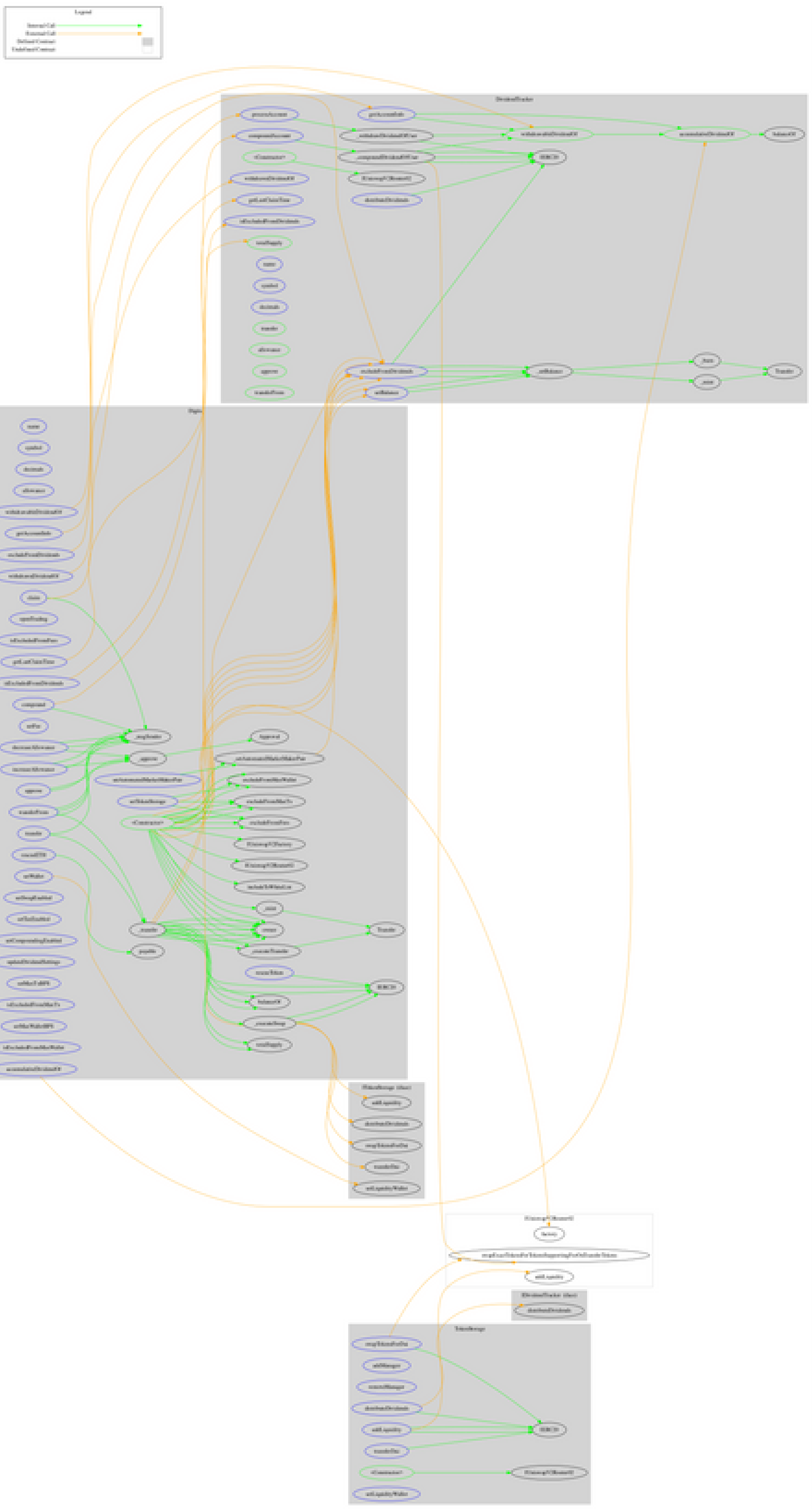
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	External		-
	_setBalance	Internal	✓	
	_mint	Private	✓	
	_burn	Private	✓	
	processAccount	External	✓	onlyOwner
	_withdrawDividendOfUser	Private	✓	
	compoundAccount	External	✓	onlyOwner
	_compoundDividendOfUser	Private	✓	
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	Public		-
	getAccountInfo	External		-
	getLastClaimTime	External		-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public		-
	allowance	Public		-
	approve	Public		-

	transferFrom	Public		-
IDividendTracker	Interface			
	distributeDividends	External	✓	-
TokenStorage	Implementation	Ownable		
		Public	✓	-
	addManager	External	✓	onlyOwner
	removeManager	External	✓	onlyOwner
	transferDai	External	✓	-
	swapTokensForDai	External	✓	-
	addLiquidity	External	✓	-
	distributeDividends	External	✓	-
	setLiquidityWallet	External	✓	-

Inheritance Graph



Flow Graph



Summary

Digits DAO contracts implement a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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