Cyberscope

Audit Report

Digits DAO

December 2022

SHA256 2f1d9d7d22c2b9ff17046cc70d4604a30dd0e4a68f59144309f2bd4f3687e38a

Audited by © cyberscope

Table of Contents

Review

Contract Name	MultiRewards
Testing Deploy	https://testnet.bscscan.com/address/0xd34a5ca5735764abc5d23dee9ccb903ba6ca4294

Audit Updates

Initial Audit	19 Dec 2022 https://github.com/cyberscope-io/audits/blob/main/digits-dao/v1/MultiRewards.pdf
Corrected Phase 2	30 Dec 2022

Source Files

Filename	SHA256
contracts/MultiRewards.sol	2f1d9d7d22c2b9ff17046cc70d4604a30dd0e4a68f59144309f2bd4f3687e38a

Introduction

Digits DAO consists of four contracts:

- Digits
- DividendTracker
- TokenStorage
- MultiRewards

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

Roles

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

The OWNER role has the authority to

- Add a new reward.
- Set the distributor of a reward.
- Pause the staking functionality.

The USER role has the authority to

- Stake tokens.
- Withdraw tokens.
- Get rewards.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	AAO	Accumulated Amount Overflow	Acknowledged
●	PAI	Potential Amount Inconsistency	Acknowledged
●	DPI	Decimals Precision Inconsistency	Acknowledged
●	IRRC	Inaccurate Reward Rate Calculation	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L09	Dead Code Elimination	Acknowledged
●	L16	Validate Variable Setters	Acknowledged
●	L17	Usage of Solidity Assembly	Acknowledged

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MultiRewards.sol#L629,641
Status	Acknowledged

Description

The transfer and transferFrom functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
stakingToken.safeTransferFrom(msg.sender, address(this), amount);
...
stakingToken.safeTransfer(msg.sender, amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has transferred instead of the expected. It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could calculate the actual amount by deducting the amount after from the amount before the transfer.

Team Update

“We treat reflection (tax) as a feature and add various contracts (like Multirewards) to allowlist to allow staking and withdrawing without tax - allowlist is in the Digits.sol contract.”

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MultiRewards.sol#L555
Status	Acknowledged

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
function rewardPerToken(address _rewardsToken) public view returns (uint256)
{
  if (_totalSupply == 0) {
    return rewardData[_rewardsToken].rewardPerTokenStored;
  }
  return
    rewardData[_rewardsToken].rewardPerTokenStored.add(
      lastTimeRewardApplicable(_rewardsToken)
        .sub(rewardData[_rewardsToken].lastUpdateTime)
        .mul(rewardData[_rewardsToken].rewardRate)
        .mul(1e18)
        .div(_totalSupply)
    );
}
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

Team Update

“We control adding more rewards which triggers setting variables for caluclations. We have one check for overflow in notifyRewardAmount() function, and will also monitor adding proper amount of rewards. However as adding rewards requires sending them to the contract, it's highely unlikely we will top it up with so much DAI and USDC (as we intend to) to ever be a problem, as it would require immense capital.”

PAI - Potential Amount Inconsistency

Criticality	Minor / Informative
Location	MultiRewards.sol#L772
Status	Acknowledged

Description

The result that is being returned from the `withdrawableDividendOf` function can potentially be different than the one being transferred at the `claim` function. As a result, the values may diverge.

```
uint256 reflection = _digits.withdrawableDividendOf(address(this));
if (reflection > 0) {
  _digits.claim();
  _reflectionPerToken = reflection.mul(1e32).div(_totalSupply).add(
    _reflectionPerToken
  );
}
```

Recommendation

The team is advised to take into consideration the actual amount that has transferred instead of the expected.

Team Update

“*We are confident that withdrawableDividendOf() value is used in claim() and processAccount() function as intended. Comment about trusting implementation was put if the code was to be used with other contracts as these needs match.*”

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	MultiRewards.sol#L555,664
Status	Acknowledged

Description

The variable `rewardRate` is calculated based on the decimal places of the `rewardToken`, when the `notifyRewardAmount` function is called. In contrast, the `rewardPerToken` function uses the `rewardRate` variable with the decimal places of the contract. As a result, this can cause inconsistency between values.

```
IERC20(_rewardsToken).safeTransferFrom(msg.sender, address(this), reward);

if (block.timestamp >= rewardData[_rewardsToken].periodFinish) {
  rewardData[_rewardsToken].rewardRate = reward.div(
    rewardData[_rewardsToken].rewardsDuration
  );
} else {
  uint256 remaining = rewardData[_rewardsToken].periodFinish.sub(
    block.timestamp
  );
  uint256 leftover = remaining.mul(
    rewardData[_rewardsToken].rewardRate
  );
  rewardData[_rewardsToken].rewardRate = reward.add(leftover).div(
    rewardData[_rewardsToken].rewardsDuration
  );
}
...
rewardData[_rewardsToken].rewardPerTokenStored.add(
  lastTimeRewardApplicable(_rewardsToken)
    .sub(rewardData[_rewardsToken].lastUpdateTime)
    .mul(rewardData[_rewardsToken].rewardRate)
    .mul(1e18)
    .div(_totalSupply)
);
```

Recommendation

To avoid these issue, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to check the difference of decimal places between the `rewardToken` and the `staking` token, after the reward is transferred on `notifyRewardAmount` function and normalize it.

Team Update

“*We are confident that in these functions different token decimals do not result in inconsistencies, or if they do they remain in acceptable threshold, which was tested by us.*”

IRRC - Inaccurate Reward Rate Calculation

Criticality	Minor / Informative
Location	MultiRewards.sol#L696
Status	Acknowledged

Description

The contract calculates the `rewardRate` based on the remaining period of a user's last staking, but it can lead to inaccurate values. An example might be the following:

- A user stakes 100 tokens, then the values are being calculated.
- Before the time period finishes, he stakes again the same amount.
- Then the `rewardRate` is being calculated again with the last's staking reward and time period, without taking into consideration the first one. As a result `rewardRate` resolves to an inaccurate number.

```
uint256 remaining = rewardData[_rewardsToken].periodFinish.sub(
  block.timestamp
);
uint256 leftover = remaining.mul(
  rewardData[_rewardsToken].rewardRate
);
rewardData[_rewardsToken].rewardRate = reward.add(leftover).div(
  rewardData[_rewardsToken].rewardsDuration
);
```

Recommendation

The team is advised to recoinsider all possible scenarios and rewrite the `notifyRewardAmount` function to avoid any potential miscalculations.

Team Update

“We are confident that the rewards are updated in updareReward() before each stake() and notifyRewardAmount() and that the rewards and reward rates are calcualted with acceptable threshold for errors, which was tested by us.”

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/MultiRewards.sol#L151,196,523,524,525,551,560,578,605,619,620,669,730
Status	Acknowledged

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.

```
address _owner
bool _paused
address _rewardsToken
address _rewardsDistributor
uint256 _rewardsDuration
```

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.

Team Update

“*Code was just repurposed, we didn’t want to make too many changes to working code, so we will leave these as is.*”

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/MultiRewards.sol#L120,135,280,299,317
Status	Acknowledged

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 max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + (((a % 2) + (b % 2)) / 2);
    }

...
```

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.

Team Update

“*Code belongs to external library, so we will leave these as is.*”

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/MultiRewards.sol#L152
Status	Acknowledged

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.

```
nominatedOwner = _owner
```

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.

Team Update

“*Code belongs to external library, so we will leave these as is.*”

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/MultiRewards.sol#L25
Status	Acknowledged

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(account)
        }
```

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.

Team Update

“*Code belongs to external library, so we will leave these as is.*”

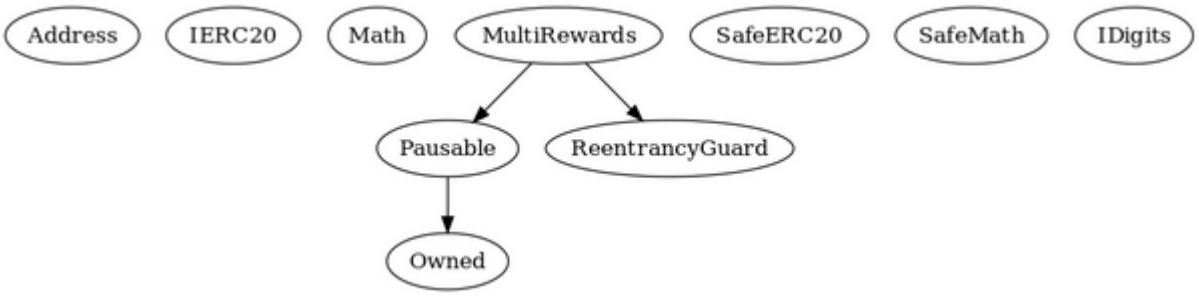
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
Owned	Implementation			
		Public	✓	-
	nominateNewOwner	External	✓	onlyOwner
	acceptOwnership	External	✓	-
	_onlyOwner	Private		
Pausable	Implementation	Owned		
		Internal	✓	

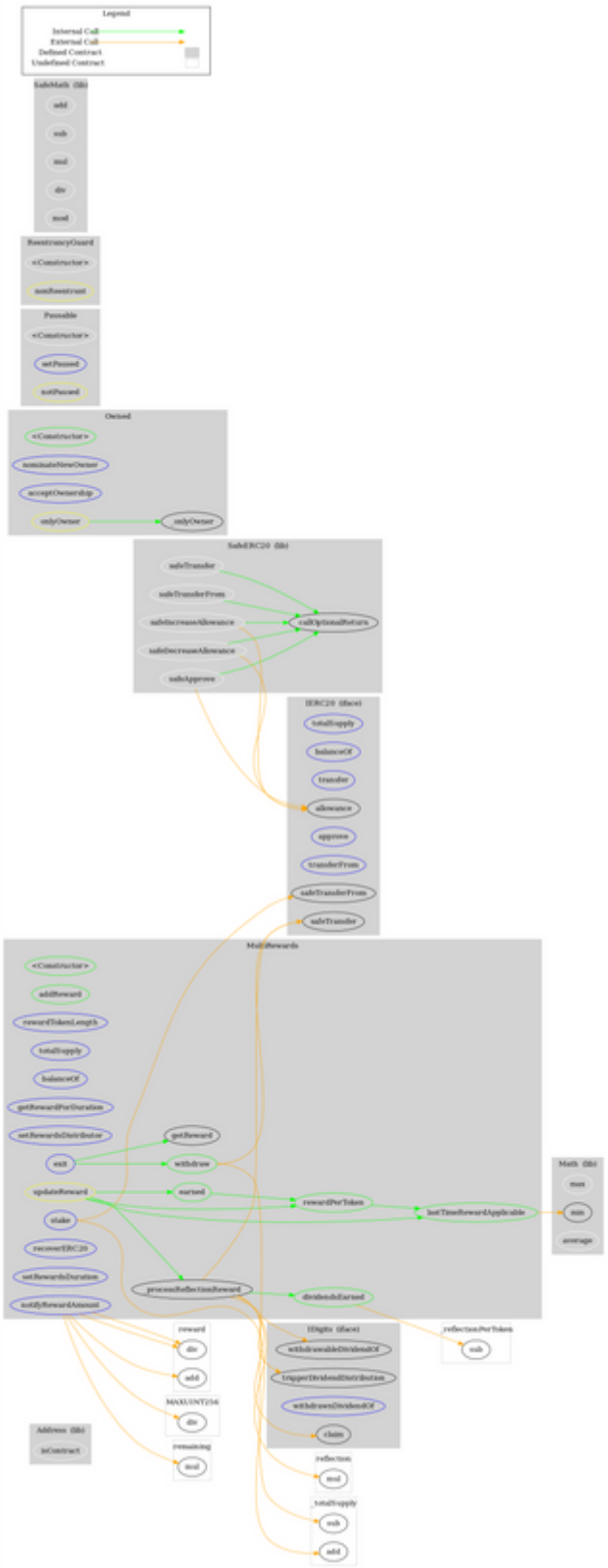
	setPaused	External	✓	onlyOwner
ReentrancyGuard	Implementation			
		Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	callOptionalReturn	Private	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
IDigits	Interface			
	claim	External	✓	-
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	triggerDividendDistribution	External	✓	-
MultiRewards	Implementation	Reentrancy Guard, Pausable		
		Public	✓	Owned

	addReward	Public	✓	onlyOwner
	rewardTokenLength	External		-
	totalSupply	External		-
	balanceOf	External		-
	lastTimeRewardApplicable	Public		-
	rewardPerToken	Public		-
	earned	Public		-
	dividendsEarned	Public		-
	getRewardForDuration	External		-
	setRewardsDistributor	External	✓	onlyOwner
	stake	External	✓	nonReentrant notPaused updateReward
	withdraw	Public	✓	nonReentrant updateReward
	getReward	Public	✓	nonReentrant updateReward
	exit	External	✓	-
	notifyRewardAmount	External	✓	updateReward
	recoverERC20	External	✓	onlyOwner
	setRewardsDuration	External	✓	-
	_processReflectionReward	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
ReentrancyGuard
<Constructor>
nonReentrant
Pausable
<Constructor>
setPaused
notPaused
Ownerd
<Constructor>
onlyOwner
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeIncreaseAllowance
safeDecreaseAllowance
safeApprove
callOptionalReturn
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
MultiRewards
<Constructor>
addReward
rewardTokensLength
totalSupply
balanceOf
getRewardForDuration
setRewardsDistributor
exit
getReward
withdraw
updateReward
earned
rewardPerToken
lastTimeRewardApplicable
stake
recoverERC20
setRewardsDuration
notifyRewardAmount
processReflectionReward
dividendsEarned
Math (lib)
max
min
average
reward
div
add
MAXUINT256
div
remaining
mul
Address (lib)
isContract
IDigits (iface)
withdrawableDividendOf
triggerDividendDistribution
withdrawnDividendOf
claim
reflection
mul
totalSupply
sub
add
reflectionPerToken
sub

Summary

Digits DAO contract implements a staking mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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