Cyberscope

Audit Report

Cactus Reward Token

February 2023

Type	BEP20
Network	BSC
Address	0x28CF95076Cc52cfB6339dadFF8150Db1A5958E55
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Roles 4
Analysis 5
MT - Mints Tokens 6
Description 6
Recommendation 7
Team Update 8
Diagnostics 9
L02 - State Variables could be Declared Constant 10
Description 10
Recommendation 10
L04 - Conformance to Solidity Naming Conventions 11
Description 11
Recommendation 11
L07 - Missing Events Arithmetic 13
Description 13
Recommendation 13
L13 - Divide before Multiply Operation 14
Description 14
Recommendation 14
L16 - Validate Variable Setters 15
Description 15
Recommendation 15
L19 - Stable Compiler Version 16
Description 16
Recommendation 16
Functions Analysis 17
Inheritance Graph 21

Flow Graph 22

Summary 23

Disclaimer 24

About Cyberscope 25

Review

Contract Name	CactusRewardToken
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x28cf95076cc52cfb6339dadff8150db1a5958e55
Address	0x28cf95076cc52cfb6339dadff8150db1a5958e55
Network	BSC
Symbol	CRT
Decimals	18
Total Supply	104.824

Audit Updates

Initial Audit	15 Feb 2023 https://github.com/cyberscope-io/audits/tree/main/4-crt/v1/audit.pdf
Corrected Phase 2	22 Feb 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Address.sol	1e0922f6c0bf6b1b8b4d480dcabb691b1359195a297bde6dc5172e79f3a1f826
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
contracts/CactusRewardToken.sol	6c213a9dcba7a062c112c2f0ad54336fe534544b557d8ddb277e01ccc8e2027f

Roles

The contract roles consist of Owner and Operator role.

The `Owner` has the authority to mint tokens.

The `Operator` has the authority to

- Change payable address.
- Grant or revoke the Operator role.
- Change uniswap pair address.

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	contracts/CactusRewardToken.sol#L62,99
Status	Acknowledged

Description

The contract's owner is a Staking contract, which means the ownership cannot be transferred and has the authority to mint tokens. The Staking contract mints tokens to the `CactusRewardToken` when a user claims the staking reward and the reward amount is greater than the token's contract balance. The max reward amount that can be claimed by a user is equal to 45.6% of the token's total supply. This is possible if a user stakes the maximum stake amount and waits for almost a year to claim the reward.

Additionally, the users have the authority to mint tokens by calling the function `claim`. As a result, the contract tokens will be highly inflated.

```
function claim() public payable returns (bool) {
  require(
      msg.value >= _airdropEth,
      "0.0176 BNB (~$5) is the minimum required to claim CRT"
  );
  require(
      msg.value <= _maxAirdropEth,
      "3.52 BNB (~$1000) is the maximum to claim CRT"
  );
  uint256 amountToMint = msg.value.div(_airdropBaseEth).mul(
      _airdropSingleToken
  );
  require(
      mintableAirdropTokens >= amountToMint,
      "There are no more tokens to claim"
  );
  if (participants[msg.sender] == address(0)) {
      numParticipants = numParticipants.add(1);
      participants[msg.sender] = msg.sender;
  }
  mintableAirdropTokens = mintableAirdropTokens.sub(amountToMint);
  fundRaised = fundRaised.add(msg.value);
  _mint(msg.sender, amountToMint);
  payable(payableAddress).transfer(msg.value);

  return true;
}

function mint(address to, uint256 amount) external onlyOwner {
  _mintTokens(to, 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.

Team Update

The team responded with the following statement:

“The Mint function of the CRT token is controlled by the staking contract. And not by any individual owner or operator. The staking contract is designed to only mint new tokens when the CRT tokens in its reward pool have been exhausted. The staking contract is the owner of the CRT token.

The operator has no function over the contract when it comes to minting function. Therefore the concern for inflation in the tokens cannot come to be. Since the staking contract is the only owner with access to mint tokens, it only does it on a measured basis.”

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L18,19,20,21,22,23,30,31,32
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 public cap = 100000000000000000000000000
uint256 public airdropCap = 80000000000000000000000000
uint256 private _airdropEth = 17505263157894736
uint256 private _maxAirdropEth = 3520000000000000000
uint256 private _airdropBaseEth = 352
uint256 private _airdropSingleToken = 400000
uint256 private _taxFeeOnBuy = 300
uint256 private _taxFeeOnSell = 300
uint256 private _bpsBase = 10000
```

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	contracts/CactusRewardToken.sol#L103,107,114
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.

```
address _payableAddress
address _uniswapV2PairAddress
address _operator
bool _status
```

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	contracts/CactusRewardToken.sol#L82
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.

```
mintableAirdropTokens = mintableAirdropTokens.sub(amountToMint)
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L71
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 amountToMint = msg.value.div(_airdropBaseEth).mul(
            _airdropSingleToken
        )
```

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	contracts/CactusRewardToken.sol#L47,104
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.

```
payableAddress = _payableAddress
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L2
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 ^0.8.0;
```

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.

Functions Analysis

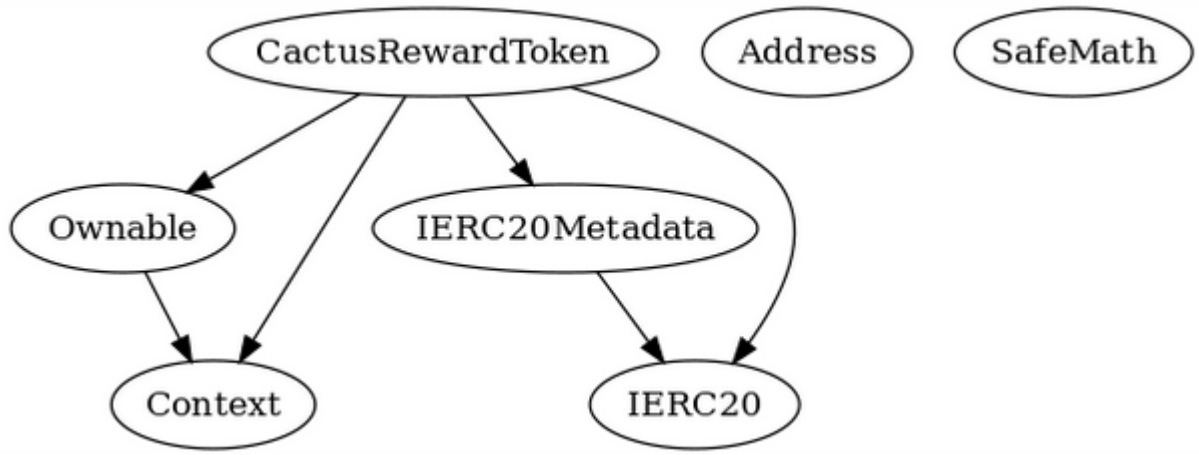
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		

	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		

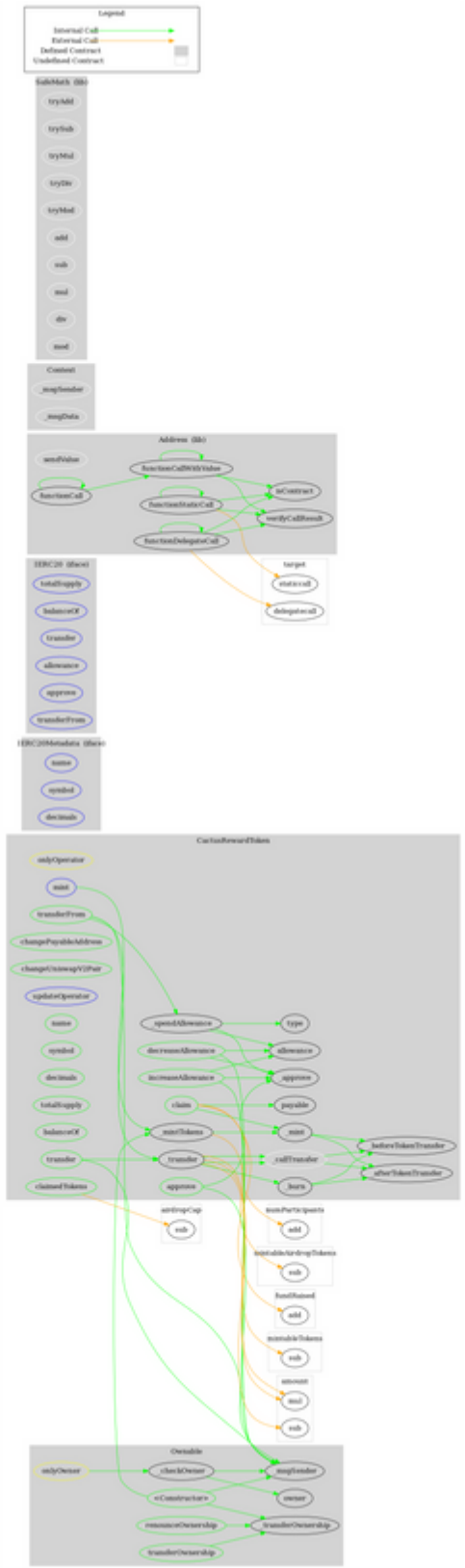
CactusReward Token	Implementation	Context, IERC20, IERC20Metadata, Ownable		
		Public	✓	-
	claimedTokens	Public		-
	claim	Public	Payable	-
	_mintTokens	Internal	✓	
	mint	External	✓	onlyOwner
	changePayableAddress	Public	✓	onlyOperator
	changeUniswapV2Pair	Public	✓	onlyOperator
	updateOperator	External	✓	onlyOperator
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_callTransfer	Internal	✓	
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph



Summary

There are some functions that can be abused by the owner like mint tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a max of 3% fee.

Regarding the mint functionality, the team responded with the following statement:

“The Mint function of the CRT token is controlled by the staking contract. And not by any individual owner or operator. The staking contract is designed to only mint new tokens when the CRT tokens in its reward pool have been exhausted. The staking contract is the owner of the CRT token.

The operator has no function over the contract when it comes to minting function. Therefore the concern for inflation in the tokens cannot come to be. Since the staking contract is the only owner with access to mint tokens, it only does it on a measured basis.”

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

Cyberscope

Audit Report

Cactus Reward Token

February 2023

Type	BEP20
Network	BSC
Address	0x28CF95076Cc52cfB6339dadFF8150Db1A5958E55
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Roles 4
Analysis 5
MT - Mints Tokens 6
Description 6
Recommendation 7
Diagnostics 8
L02 - State Variables could be Declared Constant 9
Description 9
Recommendation 9
L04 - Conformance to Solidity Naming Conventions 10
Description 10
Recommendation 10
L07 - Missing Events Arithmetic 12
Description 12
Recommendation 12
L13 - Divide before Multiply Operation 13
Description 13
Recommendation 13
L16 - Validate Variable Setters 14
Description 14
Recommendation 14
L19 - Stable Compiler Version 15
Description 15
Recommendation 15
Functions Analysis 16
Inheritance Graph 20
Flow Graph 21

Summary 22

Disclaimer 23

About Cyberscope 24

Review

Contract Name	CactusRewardToken
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x28cf95076cc52cfb6339dadff8150db1a5958e55
Address	0x28cf95076cc52cfb6339dadff8150db1a5958e55
Network	BSC
Symbol	CRT
Decimals	18
Total Supply	104.824

Audit Updates

Initial Audit	15 Feb 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Address.sol	1e0922f6c0bf6b1b8b4d480dcabb691b1359195a297bde6dc5172e79f3a1f826
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
contracts/CactusRewardToken.sol	6c213a9dcba7a062c112c2f0ad54336fe534544b557d8ddb277e01ccc8e2027f

Roles

The contract roles consist of Owner and Operator role.

The `Owner` has the authority to mint tokens.

The `Operator` has the authority to

- Change payable address.
- Grant or revoke the Operator role.
- Change uniswap pair address.

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	contracts/CactusRewardToken.sol#L62,99
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. Additionally, the users have the authority o mint tokens by calling the function `claim`. As a result, the contract tokens will be highly inflated.

```
function claim() public payable returns (bool) {
  require(
      msg.value >= _airdropEth,
      "0.0176 BNB (~$5) is the minimum required to claim CRT"
  );
  require(
      msg.value <= _maxAirdropEth,
      "3.52 BNB (~$1000) is the maximum to claim CRT"
  );
  uint256 amountToMint = msg.value.div(_airdropBaseEth).mul(
      _airdropSingleToken
  );
  require(
      mintableAirdropTokens >= amountToMint,
      "There are no more tokens to claim"
  );
  if (participants[msg.sender] == address(0)) {
      numParticipants = numParticipants.add(1);
      participants[msg.sender] = msg.sender;
  }
  mintableAirdropTokens = mintableAirdropTokens.sub(amountToMint);
  fundRaised = fundRaised.add(msg.value);
  _mint(msg.sender, amountToMint);
  payable(payableAddress).transfer(msg.value);

  return true;
}

function mint(address to, uint256 amount) external onlyOwner {
  _mintTokens(to, 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.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L18,19,20,21,22,23,30,31,32
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 public cap = 100000000000000000000000000
uint256 public airdropCap = 80000000000000000000000000
uint256 private _airdropEth = 17505263157894736
uint256 private _maxAirdropEth = 3520000000000000000
uint256 private _airdropBaseEth = 352
uint256 private _airdropSingleToken = 400000
uint256 private _taxFeeOnBuy = 300
uint256 private _taxFeeOnSell = 300
uint256 private _bpsBase = 10000
```

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	contracts/CactusRewardToken.sol#L103,107,114
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.

```
address _payableAddress
address _uniswapV2PairAddress
address _operator
bool _status
```

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	contracts/CactusRewardToken.sol#L82
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.

```
mintableAirdropTokens = mintableAirdropTokens.sub(amountToMint)
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L71
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 amountToMint = msg.value.div(_airdropBaseEth).mul(
            _airdropSingleToken
        )
```

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	contracts/CactusRewardToken.sol#L47,104
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.

```
payableAddress = _payableAddress
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/CactusRewardToken.sol#L2
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 ^0.8.0;
```

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.

Functions Analysis

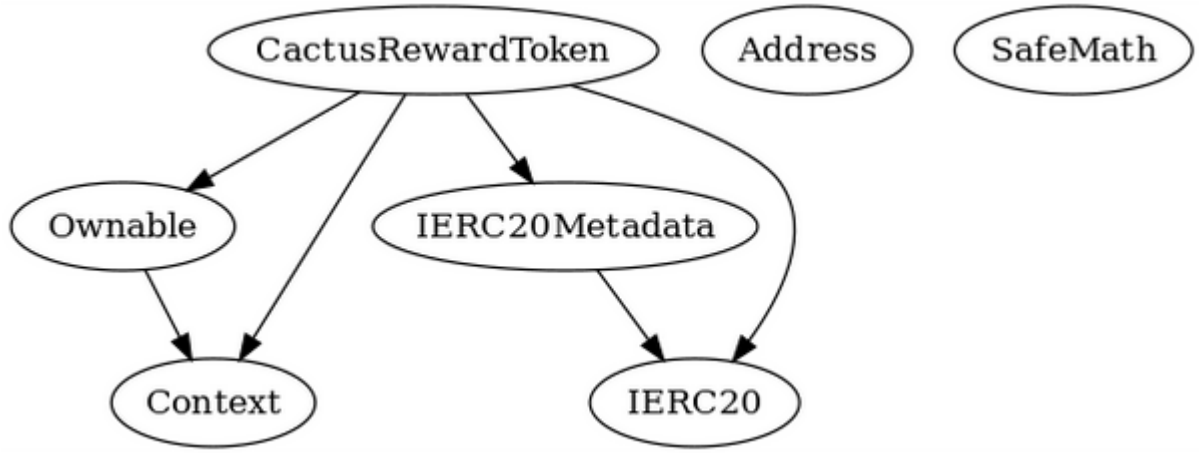
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		

	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		

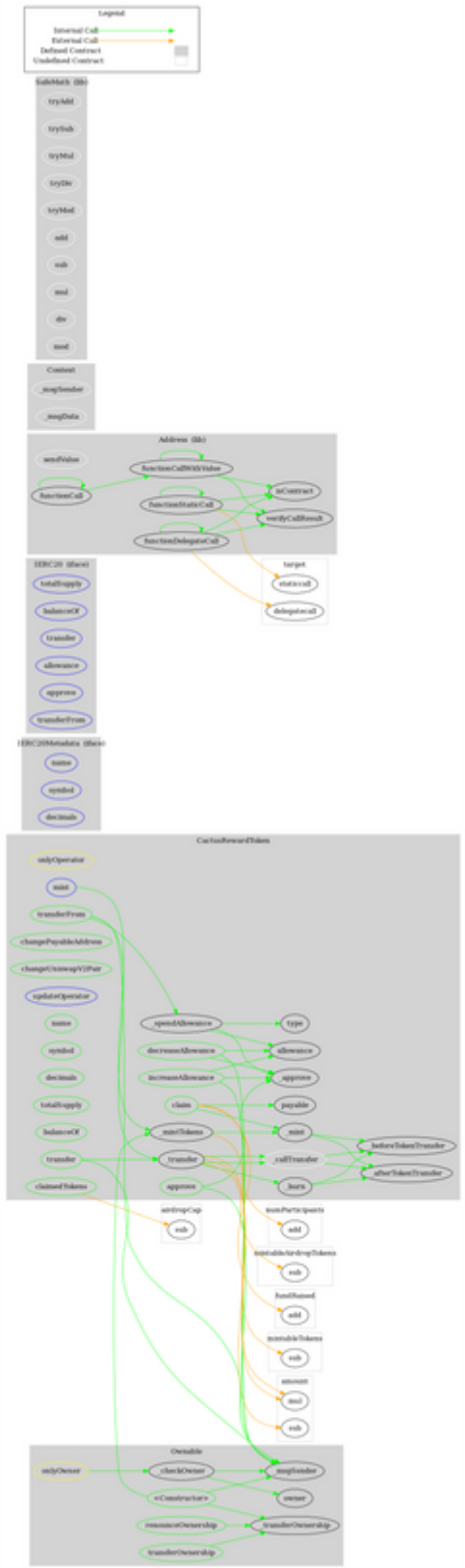
CactusReward Token	Implementation	Context, IERC20, IERC20Metadata, Ownable		
		Public	✓	-
	claimedTokens	Public		-
	claim	Public	Payable	-
	_mintTokens	Internal	✓	
	mint	External	✓	onlyOwner
	changePayableAddress	Public	✓	onlyOperator
	changeUniswapV2Pair	Public	✓	onlyOperator
	updateOperator	External	✓	onlyOperator
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_callTransfer	Internal	✓	
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph



Summary

There are some functions that can be abused by the owner like mint tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a max of 3% fee.

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