Cyberscope

Audit Report

MINGMONG

July 2023

SHA256 379a9506a5c10cb04585a7dcce56fe6ebb4ce4ce53a60369b42584da758068ff

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0xfd7671cd438ef91ab797bde0c910d7123471e7e6

Audit Updates

Initial Audit	02 Jul 2023

Source Files

Filename	SHA256
contracts/main.sol	379a9506a5c10cb04585a7dcce56fe6ebb4ce4ce53a60369b42584da758068ff

Introduction

The MINGMONG implements a staking mechanism. Users can deposit tokens to a pool and receive rewards in the form of another token. The contract includes multiple pools, each with its own deposit and withdrawal fees, minimum deposit amount, and harvest interval. The contract also includes a mechanism for locking up rewards and penalizing early withdrawals. The contract owner can add and update pools, start and pause the staking platform, and emergency withdraw tokens from any pool.

Roles

Owner

The owner has authority over the following functions:

- `function initialize()`
- `function add(uint256 _tokenPerBlock, IERC20 _stakedToken, IERC20 _rewardToken, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval, WithdrawFeeInterval[] memory withdrawFeeIntervals)`
- `function set(uint256 _pid, uint256 _tokenPerBlock, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval)`
- `function setWithdrawFeeInterval(uint256 poolId, uint256 index, WithdrawFeeInterval memory _withdrawFeeInterval)`
- `function emergencyAdminWithdraw(uint256 _pid)`
- `function updatePaused(bool _value)`
- `function setLockDeposit(uint256 pid, bool locked)`

User

The user can interact with the following functions:

- `function getWithdrawFeeIntervals(uint256 poolId)`
- `function poolLength()`
- `function getMultiplier(uint256 _from, uint256 _to)`
- `function pendingToken(uint256 _pid, address _user)`
- `function canHarvest(uint256 _pid, address _user)`
- `function depositRewardToken(uint256 poolId, uint256 amount)`
- `function deposit(uint256 _pid, uint256 _amount)`
- `function withdraw(uint256 _pid, uint256 _amount)`
- `function getWithdrawFee(uint256 poolId, uint256 stakedTime)`

- `function emergencyWithdraw(uint256 _pid)`

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	NR	Non-Guaranteed Reward	Unresolved
●	GO	Gas Optimization	Unresolved
●	RV	Redundant Variable	Unresolved
●	RI	Reward Inconsistency	Unresolved
●	SS	State Synchronization	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/main.sol#L1002,1106,1115
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 `balanceOf` function of an ERC20 token returns an `uint256` number, which means its value will always be greater than or equal to zero. The contract checks if its balance is greater than zero. This operation can be improved by using the `!=` operator.

```
pool.rewardToken.balanceOf(address(this)) > 0
pending > 0 || user.rewardLockedUp > 0
```

Recommendation

The team is advised to take these segments into consideration 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. As described above, it is recommended to use the `!=` operator.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	contracts/main.sol#L1087
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

The `_withdrawFee` must be less than or equal to 1000 for the subtraction to work as expected. The owner has the authority to alter the fees of a certain poolId, and the `setWithdrawFeeInterval` function has no input validation. As a result, the operation may lead to an underflow.

```
uint256 _withdrawFee = getWithdrawFee(_pid, user.depositTimestamp);
uint256 feeAmount = _amount.mul(_withdrawFee).div(1000);
amountToTransfer = _amount.sub(feeAmount);
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

NR - Non-Guaranteed Reward

Criticality	Minor / Informative
Location	contracts/main.sol#L1141
Status	Unresolved

Description

The contract does not guarantee the users will get their rewards. The deposited amounts are transferred to the contract's balance, but the claimable rewards are transferred to the users from the `Reserve` contract. Additionally, the owner has the authority to claim all the reward funds from the `Reserve` contract.

```
function emergencyAdminWithdraw(uint256 _pid) external onlyOwner {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 balanceToWithdraw = pool.rewardToken.balanceOf(address(this));
    require(balanceToWithdraw != 0, "STAKING: Not enough balance to
withdraw!");
    pool.rewardToken.transfer(owner(), balanceToWithdraw);
    rewardReserve.safeTransfer(pool.rewardToken, owner(),
pool.rewardToken.balanceOf(address(rewardReserve)));
    emit AdminEmergencyWithdraw(_pid,
pool.rewardToken.balanceOf(address(this)), pool.accTokenPerShare,
pool.tokenPerBlock, pool.lastRewardBlock);
    delete poolInfo[_pid];
}
```

Recommendation

The contract could guarantee the rewards will be available when a user deposits the amount. The owner is responsible for the `Reserve` contract having all the available funds so that users can claim their reward. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/main.sol#L875
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 `MAXIMUM_HARVEST_INTERVAL` and `harvestInterval` are declared as `uint256` variables. The `MAXIMUM_HARVEST_INTERVAL` is declared as a constant, hence its value cannot change. 14 days is equivalent to `14 * 24 * 60 * 60` seconds. This value can be stored in an `uint32` variable. Storing these variables as `uint256` takes up a lot more storage space, especially in the case of `harvestInterval`, which is a struct property stored in an array. As a result, the contract takes up more storage space than it actually requires, and has increased gas costs for transactions that include this variable.

```
uint256 public constant MAXIMUM_HARVEST_INTERVAL = 14 days;
```

Recommendation

The team is advised to take these segments into consideration 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.

RV - Redundant Variable

Criticality	Minor / Informative
Location	contracts/main.sol#L874
Status	Unresolved

Description

The `BONUS_MULTIPLIER` is a constant variable with a value of 1. It is only used for multiplication, hence the variable is redundant.

```
uint256 public constant BONUS_MULTIPLIER = 1;
```

Recommendation

The team is advised to remove this variable from the contract as it is redundant.

RI - Reward Inconsistency

Criticality	Minor / Informative
Location	contracts/main.sol#L828
Status	Unresolved

Description

The reward mechanism of the platform is not consistent. The `safeTransfer` function compares the reward amount with the Reserve contract's balance. If the amount is greater, then the user will receive a reward equal to the contract's balance. Since the owner has the authority to claim the balance of the Reserve contract, a user can claim the reward with the possibility of the reward amount the user receives is zero, if the owner previously claimed all the funds. The claiming process will be considered successful though. As a result, the reward mechanism is inconsistent.

```
function safeTransfer(
    IERC20 rewardToken,
    address _to,
    uint256 _amount
) external onlyOwner {
    uint256 tokenBal = rewardToken.balanceOf(address(this));
    if (_amount > tokenBal) {
        rewardToken.transfer(_to, tokenBal);
    } else {
        rewardToken.transfer(_to, _amount);
    }
}
```

Recommendation

The owner is responsible for the Reserve contract having all the available funds so that users can claim their reward. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

SS - State Synchronization

Criticality	Minor / Informative
Location	contracts/main.sol#L995
Status	Unresolved

Description

The contract's state is not being synchronized with what the user sees and the actual values. The `pendingToken` function will return the reward the user is actually entitled to, but in reality, the reward may be much less or even zero. This is a consequence of the RI and NR findings, which are described in detail below.

```
function pendingToken(uint256 _pid, address _user) external view returns
(uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accTokenPerShare = pool.accTokenPerShare;
    if (
        block.number > pool.lastRewardBlock &&
        pool.stakedAmount != 0 &&
        pool.rewardToken.balanceOf(address(this)) > 0
    ) {
        uint256 multiplier = getMultiplier(pool.lastRewardBlock,
block.number);
        uint256 tokenReward = multiplier.mul(pool.tokenPerBlock);
        accTokenPerShare =
accTokenPerShare.add(tokenReward.mul(1e12).div(pool.stakedAmount));
    }
    return user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
}
```

Recommendation

The team is advised to consider all possible outcomes of the reward mechanism and provide realistic data to the users.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/main.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.

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

Recommendation

The team is advised to remove the SafeMath library. 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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/main.sol#L996,997,1012,1123
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
PoolInfo storage pool
UserInfo storage user
WithdrawFeeInterval[] storage _withdrawFee
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/main.sol#L899
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
rewardReserve
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/main.sol#L830,831,928,929,930,931,932,933,967,968,969,970,971,984,991,995,1011,1016,1048,1076,1096,1131,1141,1151
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 _to
uint256 _amount
uint256 _tokenPerBlock
IERC20 _stakedToken
IERC20 _rewardToken
uint16 _depositFeeBP
uint256 _minDeposit
uint256 _harvestInterval
uint256 _pid
WithdrawFeeInterval memory _withdrawFeeInterval
uint256 _to
uint256 _from
address _user
bool _value
```

Recommendation

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

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/main.sol#L393,418,447,480,490,507,517,598,614,623
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }

function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/main.sol#L371,546
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/main.sol#L2,102,112,342,560,660,687,764,826
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/main.sol#L835,837,1145
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.

```
rewardToken.transfer(_to, tokenBal)
rewardToken.transfer(_to, _amount)
pool.rewardToken.transfer(owner(), balanceToWithdraw)
```

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**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Mintable	Interface	IERC20		
	mint	External	✓	-
	transferOwnership	External	✓	-

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		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	

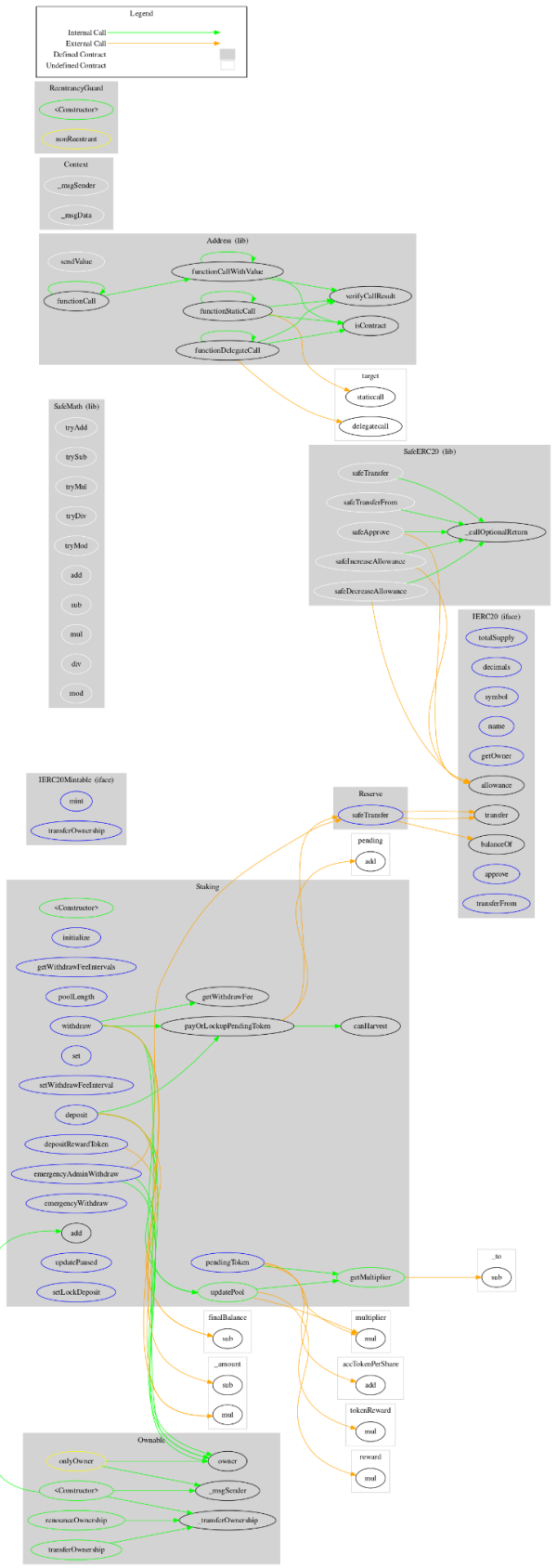
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
Reserve	Implementation	Ownable		
	safeTransfer	External	✓	onlyOwner
Staking	Implementation	Ownable		
		Public	✓	-
	initialize	External	✓	onlyOwner

	getWithdrawFeeIntervals	External		-
	poolLength	External		-
	add	Public	✓	onlyOwner
	set	External	✓	onlyOwner
	setWithdrawFeeInterval	External	✓	onlyOwner
	getMultiplier	Public		-
	pendingToken	External		-
	canHarvest	Public		-
	updatePool	Public	✓	-
	depositRewardToken	External	✓	-
	deposit	External	✓	-
	withdraw	External	✓	-
	payOrLockupPendingToken	Internal	✓	
	getWithdrawFee	Public		-
	emergencyWithdraw	External	✓	-
	emergencyAdminWithdraw	External	✓	onlyOwner
	updatePaused	External	✓	onlyOwner
	setLockDeposit	External	✓	onlyOwner

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
Context
_msgSender
_msgData
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
functionDelegateCall
verifyCallResult
isContract
target
staticcall
delegatecall
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
_callOptionalReturn
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
allowance
transfer
balanceOf
approve
transferFrom
IERC20Mintable (iface)
mint
transferOwnership
Reserve
safeTransfer
pending
add
Staking
<Constructor>
initialize
getWithdrawFeeIntervals
poolLength
withdraw
set
setWithdrawFeeInterval
deposit
depositRewardToken
emergencyAdminWithdraw
emergencyWithdraw
add
updatePaused
setLockDeposit
getWithdrawFee
payOrLockupPendingToken
canHarvest
pendingToken
updatePool
getMultiplier
_to
sub
finalBalance
sub
_amount
sub
mul
multiplier
mul
accTokenPerShare
add
tokenReward
mul
reward
mul
Ownable
onlyOwner
<Constructor>
renounceOwnership
transferOwnership
owner
_msgSender
transferOwnership

Summary

MINGMONG contract implements a token 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