Cyberscope

Audit Report

MuQuant Lock

June 2023

Lock	7d5b4bccdae44185724895c0d6a22bc32f18837fcd7dc4a346071efb3ee35097
Audited by	© cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x2A36B8D4d3124fda4643ABb4B7844816894Db408

Audit Updates

Initial Audit	13 Jun 2023

Source Files

Filename	SHA256
interfaces/IAffiliate.sol	e521b3e28fd5de81295f416da6c789dbb21323347743f40eddd8979594a7cdbb
interfaces/ICRC20.sol	9e84c467c9645e8e02580b98bd13d2f1bf22f70135e52a323ffb0adef2f06e7d
interfaces/ILaunchpad.sol	0472b8fc71cdcd0c3e62611e6240b05750aa5c5a286a5b0ebf327acbda96d9cf
interfaces/IRouter.sol	4296c42fc8140930e20e017d7bcca471b43f55831e2e4864c280c578453ce730
interfaces/IVault.sol	f2ad0a441b67ec2f02633053ab17a14b6add3d5968bc099c78b98fd129418515
Launchpad.sol	334d84894df2302fb70cec522b63e5f89564701b1fc1c231821f1e1c030ce353
libraries/MuLibrary.sol	5c892a04fe9c458262bc052f303c0dc67c130b79a6557854c3797df7edd5e8c5
Lock.sol	c64a59996b4d62e3f47eb3e3bbca9d8c0e3fa76f894f93772dff37c1b5b30694

Overview

The lock contract operates as staking mechanism. The users can vest tokens by calling `lock()` method, receive rewards proportional to the time and unlock their tokens.

Roles

- Launchpad, configures the launchpad address.
- Token creator, configures the lock rules for the created token.
- Router, configures the router address.
- Vault, configures the vault address.

Any user can execute the methods lock, unlock and claim for the active tokens. When a user executes the lock method, the deposited amount of tokens is transferred to the vault. The unlock and claim methods transfer the tokens to the locker address.

Audit Comment

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Test Deployment

Contract	Explorer
Lock	https://testnet.bscscan.com/address/0x2A36B8D4d3124fda4643ABb4B7844816894Db408

Findings Breakdown

●	Critical	0
●	Medium	0
●	Minor / Informative	11

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MRC	Misleading Reward Calculation	Acknowledged
●	PSU	Potential Subtraction Underflow	Unresolved
●	MU	Modifiers Usage	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	DSV	Duplicate State Variables	Unresolved
●	EIS	Excessively Integer Size	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MC	Missing Check	Unresolved
●	UV	Unused Variables	Unresolved
●	MVN	Misleading Variables Naming	Unresolved

MRC - Misleading Reward Calculation

Criticality	Minor / Informative
Status	Acknowledged

Description

The variable `boxs[locker][token].recentTime` that is involved in the calculation of rewards is not properly initialized or assigned a default value. As a result, if this variable is left uninitialized, it can lead to unexpected behavior and potentially generate a substantial reward amount that exceeds the intended limits or expectations.

The consequence of an uninitialized variable in reward calculations can be severe, as it may allow unauthorized or unintended individuals to exploit the contract and receive an undeserved and disproportionately high reward. This not only poses a financial risk but also undermines the fairness and integrity of the contract and its intended reward distribution mechanism.

```
uint256 recent = boxs[locker][token].recentTime;
...
uint256 times = end - recent;
...
uint256 times = uint256(block.timestamp) - recent;
reward +=
    (times *
        amounts *
        configs[token].apr[
            MuLibrary.calcRange(
                amounts,
                configs[token].range
            ) - 1
        ]) /
    DENTIME;
```

Recommendation

To address this finding and mitigate the associated risks, it is crucial to ensure that the variable responsible for calculating rewards is properly initialized or assigned an appropriate default value. Initialization should occur at the appropriate stage of the contract's execution to ensure accurate and controlled reward calculations.

Furthermore, it is recommended to conduct a comprehensive review of the contract's reward calculation mechanism to ensure that all relevant variables and parameters are correctly initialized, taking into account the contract's specific requirements and intended reward distribution logic.

Team Update

The team replied with the following statement:

"*Initially, recentTime is not initialized, that is correct. But it should be noted that, in the call to the Lock function or the Claim function the first time the user calls the function, it has already been initialized. Because every time the Lock function is called, it calculates the reward of the previous period and will reattach the recentTime with the latest time. Meaning, in the first Lock, it was initialized*"

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	Lock.sol#L321
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 number in the variable `de` is the result of an external method call. The contract does not guarantee from the business logic that the `maxReward` will be greater than the `received`. Since `affiliate` is an external source, it may produce an unexpected value, as a result the received may be greater than the `maxReward`.

```
if (configs[token].affiliate != address(0)) {
    (s, b, t, r) = IAffiliate(configs[token].affiliate).calc(locker);
    received += r;
}
uint256 maxReward = (xamounts * configs[token].maxReward[rg - 1]) /
    DENOMINATOR;
uint256 de = reward + (s + b + t);
if (received + de > maxReward) {
    reset = true;
    uint256 re = maxReward - received;
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	Lock.sol#L151,152,191,192
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(usageRate <= DENOMINATOR, "Lock: usageRate");
require(
    usageAddr != address(0) && usageAddr != address(this),
    "Lock: usageAddr"
);
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

MEE - Missing Events Emission

Criticality	Minor / Informative
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

The `Locker` contract are missing fundamental event emissions. Some methods that are included in the event emmision are:

- lock
- unlock
- claim

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

DSV - Duplicate State Variables

Criticality	Minor / Informative
Location	Lock.sol#44
Status	Unresolved

Description

The contract contains multiple instances of similar state variables that serve the same purpose but are maintained separately. This redundancy can lead to increased gas consumption, as each instance requires storage allocation and maintenance during contract execution. Additionally, it necessitates additional code logic to ensure the synchronization and consistency of these separate variables, which further complicates the contract implementation.

```
address public Vault;
IVault public vault;
address public Router;
IRouter public router;
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to consolidate the repeated state variables into a single instance. By creating a single instance, the contract will reduce the gas consumption associated with storage allocation and streamline the codebase by eliminating redundant variable synchronization logic.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	Lock.sol#L51
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Since the `usageRate` should be less than 10000, then it could be stored in a `Math.log2(10000) = 13.28 -> uint16` variable.

```
DENOMINATOR = 10000
require(usageRate <= DENOMINATOR, "Lock: usageRate");
```

The timestamp in solidity represents the seconds that have elapsed from the Unix epoch period. In Solidity, this variable is `uint256`. Hence, casting the `block.timestamp` to `uint256` is a redundant operation. Additionally, assuming that the year 5000 is an upper boundary, then the `log2(10000000000) = 33.21 -> uint40`.

```
startTime
..
start
end
endVest
recent
..
recentTime
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

RC - Repetitive Calculations

Criticality	Minor / Informative
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
require(launchpad.getStable(token) != address(0), "Lock: stable");
..
configs[token] = Config(
    launchpad.getStable(token),
    startTime,
    range,
    apr,
    lockTime,
    dustTime,
    maxReward,
    vesting,
    stable ? launchpad.getStable(token) : token,
    affiliate,
    usageRate,
    usageAddr
);
..
Timeline(
    amount,
    uint256(block.timestamp),
    uint256(block.timestamp) + getEnd(token, amount),
    uint256(block.timestamp) + getEndVest(token, amount),
    uint256(block.timestamp) + getEnd(token, amount),
    0
)
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Lock.sol#L445
Status	Unresolved

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

```
boxs[locker][token].amounts += amount;

SafeERC20Upgradeable.safeTransferFrom(
    IERC20Upgradeable(launchpad.getStable(token)),
    msg.sender,
    Vault,
    amount
);
```

Recommendation

The team is advised to take into consideration the actual amount that has been 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 produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

MC - Missing Check

Criticality	Minor / Informative
Location	Locker.sol#L500
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The parameters `from` and `to` are used as indexes to the `boxs[locker][token].map` structure. Since they represent the first and last index, the parameter `to` should be greater than the parameter `from`.

```
function unlock(
    address token,
    address locker,
    uint256 from,
    uint256 to
) public virtual started(token) returns (uint256, uint256) {...}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

UV - Unused Variables

Criticality	Minor / Informative
Status	Unresolved

Description

The contract uses certain variables are assigned values within the methods but are never referenced or used in any subsequent computations, conditionals, or data manipulations. This redundancy creates unnecessary gas consumption, incurs costs for variable allocation and operations performed on unused variables. The presence of unused variables in the methods contributes to increased gas consumption, which can have a significant impact on transaction costs, scalability, and overall contract performance.

```
if (amounts == 0) {
    recent = uint256(block.timestamp);
    tick = i + 1;
    break;
}
..
recent = uint256(block.timestamp);
tick = i;
break;
```

Recommendation

To address this finding and improve both the gas performance and codebase of the contract, it is crucial to identify and remove the unused variables from the methods. A comprehensive review of the methods should be conducted to identify these variables and assess their impact on gas consumption.

Removing the unused variables will lead to improved gas efficiency, reducing the gas costs associated with storage allocation and operations on redundant variables. This optimization will result in lower transaction costs for users and enhanced scalability of the contract, allowing more efficient usage of the resources.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

```
(s, b, t, r) = IAffiliate(configs[token].affiliate).calc(locker);
...
uint256 re = maxReward - received;
reward = (reward * re) / de;
s = (s * re) / de;
b = (b * re) / de;
t = (t * re) / de;
```

Recommendation

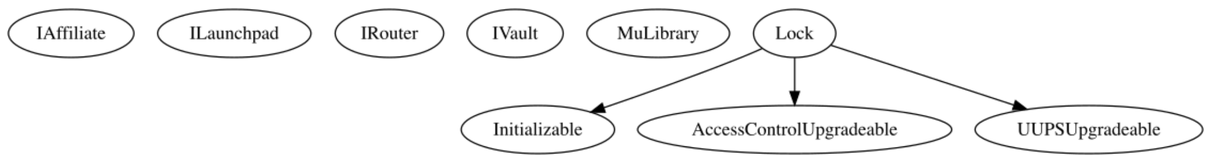
It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

Functions Analysis

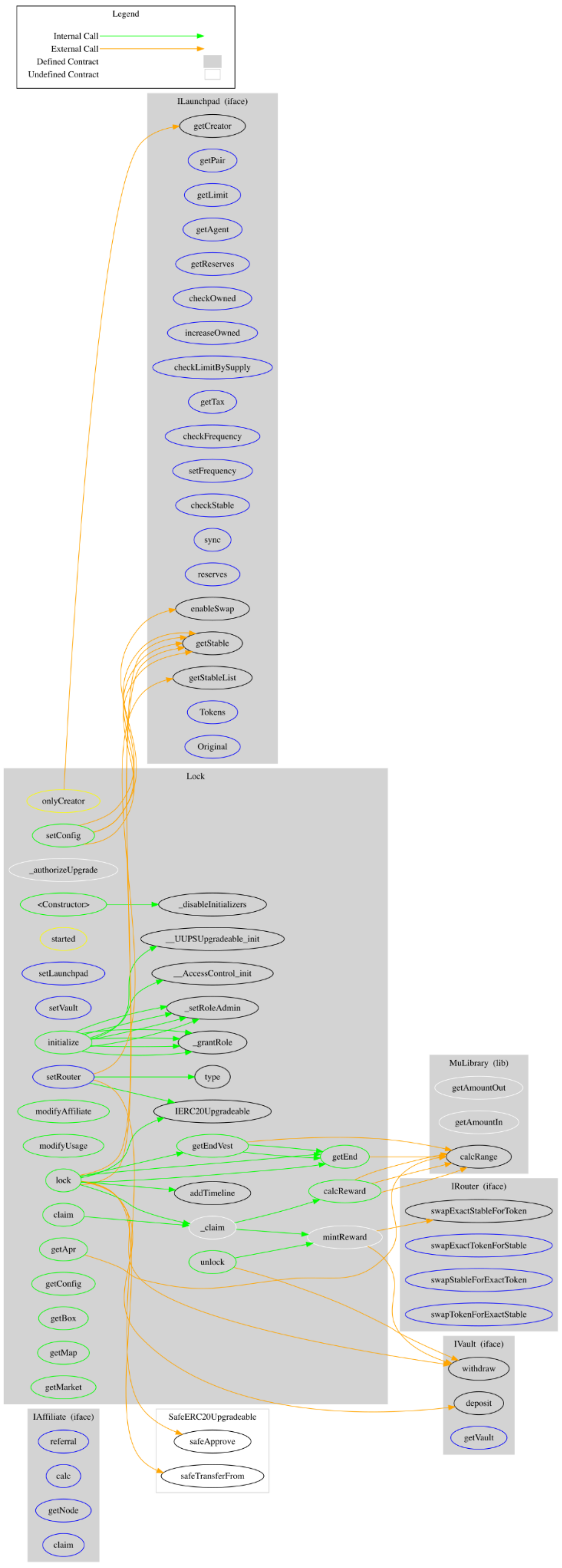
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Lock	Implementation	Initializable, AccessControlUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyRole
	setLaunchpad	External	✓	onlyRole
	setVault	External	✓	onlyRole
	setRouter	External	✓	onlyRole
	setConfig	Public	✓	onlyCreator
	modifyAffiliate	Public	✓	onlyCreator
	modifyUsage	Public	✓	onlyCreator
	getApr	Public		-
	getEnd	Public		-
	getEndVest	Public		-
	addTimeline	Internal	✓	
	calcReward	Public		-
	mintReward	Internal	✓	

	_claim	Internal	✓	started
	lock	Public	✓	started
	claim	Public	✓	started
	unlock	Public	✓	started
	getConfig	Public		-
	getBox	Public		-
	getMap	Public		-
	getMarket	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ILaunchpad (iface)
getCreator
getPair
getLimit
getAgent
getReserves
checkOwned
increaseOwned
checkLimitBySupply
getTax
checkFrequency
setFrequency
checkStable
sync
reserves
enableSwap
getStable
getStableList
Tokens
Original
Lock
onlyCreator
setConfig
_authorizeUpgrade
<Constructor>
_disableInitializers
started
__UUPSUpgradeable_init
setLaunchpad
__AccessControl_init
setVault
_setRoleAdmin
initialize
_grantRole
setRouter
type
modifyAffiliate
IERC20Upgradeable
modifyUsage
getEndVest
getEnd
lock
addTimeline
calcReward
claim
_claim
mintReward
getApr
unlock
getConfig
getBox
getMap
getMarket
MuLibrary (lib)
getAmountOut
getAmountIn
calcRange
IRouter (iface)
swapExactStableForToken
swapExactTokenForStable
swapStableForExactToken
swapTokenForExactStable
IVault (iface)
withdraw
deposit
getVault
IAffiliate (iface)
referral
calc
getNode
claim
SafeERC20Upgradeable
safeApprove
safeTransferFrom

Summary

MuQuant Lock contract implements a staking 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