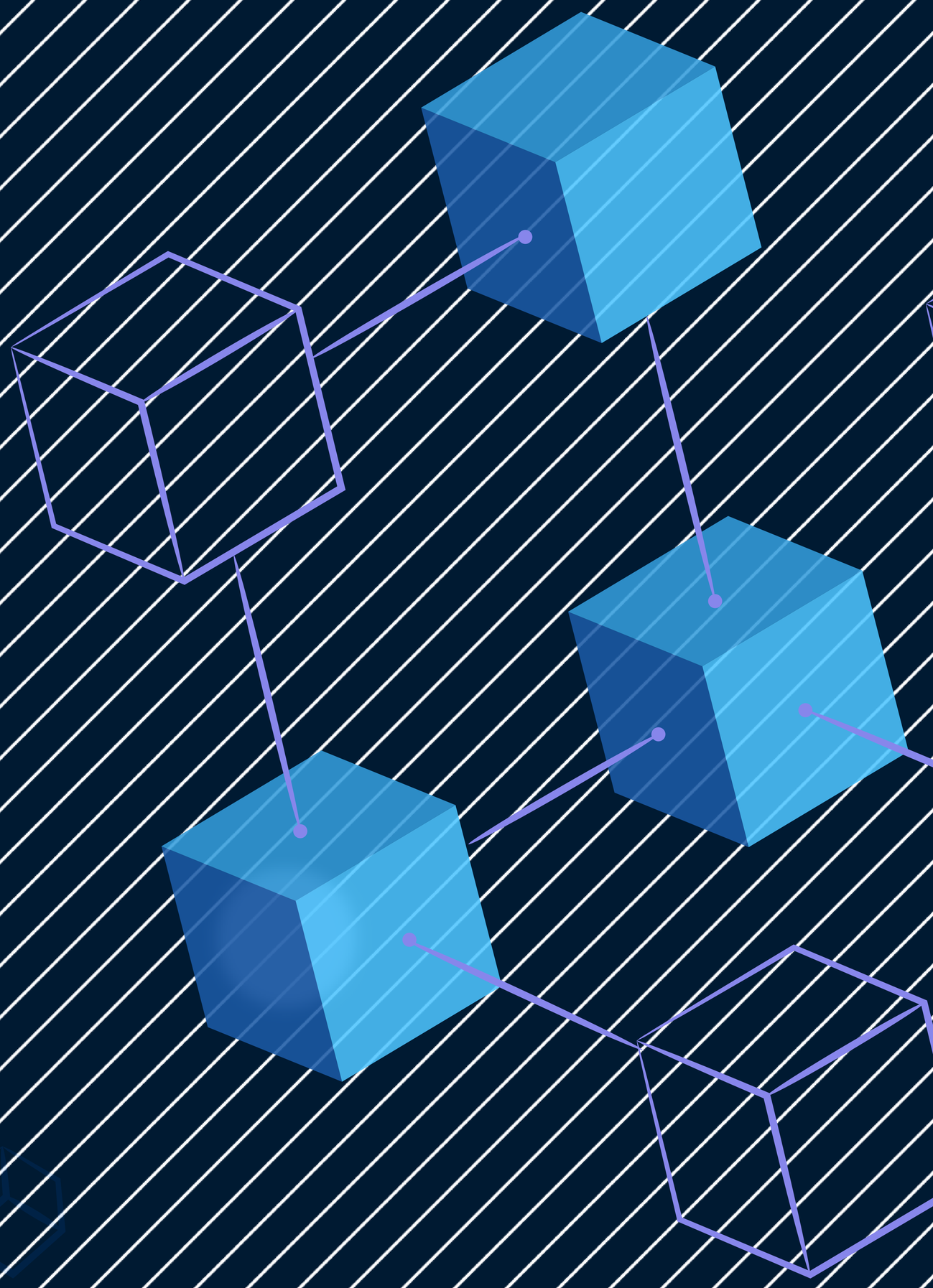
QuillAudits

# Audit Report
# October, 2021



For

alium
finance

# Contents

# Contents

## Scope of the Audit

The scope of this audit was to analyze and document the Alium smart contract codebase for quality, security, and correctness.

## Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

# Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

**Structural Analysis**
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

**Static Analysis**
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

**Code Review / Manual Analysis**
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

**Gas Consumption**
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

**Tools and Platforms used for Audit**
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

# Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

| Risk-level | Description |
| --- | --- |
| High | A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment. |
| Medium | The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed. |
| Low | Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future. |
| Informational | These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact. |

## Number of issues per severity

| Type | High | Medium | Low | Informational |
| --- | --- | --- | --- | --- |
| Open | 0 | 0 | 0 | 0 |
| Acknowledged | 0 | 0 | 1 | 0 |
| Closed | 0 | 2 | 3 | 5 |

# Introduction

During the period of **October 04, 2021, to October 07, 2021** - QuillAudits Team performed a security audit for **Alium** smart contracts.

The code for the audit was taken from following the official link: https://github.com/Alium-Finance/alium-farm/tree/master/contracts

| Ver. No. | Date | Commit hash | Files |
|---|---|---|---|
| 01 | October 4 | 545b2e29b253dbe0937<br>5d57575d9ed6ac2503145 | - MasterChef.sol<br>- FarmingTicketWindow.sol |
| 02 | October 8 | e37d6af39af68049c268<br>4085f025385407b4bd55 | - MasterChef.sol<br>- FarmingTicketWindow.sol |

# Issues Found

## A. Contract – FarmingTicketWindow

### High severity issues

No issues were found.

### Medium severity issues

#### A.1 Centralization Risks

**Description**

The role owner has the authority to:

- give the ticket to any account without any payment requirement

**Remediation**

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

- Time-lock with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

**Status:** Fixed

**Comments from Auditee:** “Founders have a Multisig with Timelock solution in place.”

## Low severity issues

### A.2 passFree() fails if the owner has the ticket

| Line | Code |
| --- | --- |
| 44-47 | function passFree(address _account) external onlyOwner notTicketHolder<br>{<br>hasTicket[_account] = true;<br>emit EntranceAllowed(_account);<br>} |

**Description**

The passFee() function uses the modifier notTicketHolder, which checks if the msg.sender is a ticket holder or not. If the msg.sender is not a ticket holder, then the execution reverts.

As this function also has the onlyOwner modifier, this means the owner will always be the msg.sender. This means if the owner has the ticket, then it can’t use the passFree() function.

**Remediation**

We recommend that instead of using the modifier, use a require statement inside the function to check for the ticket of _account.

**Status:** Fixed

In version 2, the team fixed the issue with the recommended changes.

## Informational issues

### A.3 Conformance to Solidity naming conventions

| Line | Code |
| --- | --- |
| 12 | uint256 public constant ticketPrice = 1500e18; // 1500 ALM |

**Description**
Constants should be named with all capital letters with underscores separating words. Examples: TICKET_PRICE, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION

**Remediation**
Follow the Solidity naming convention.

**Status:** Fixed
In version 2, the variable name was changed.

# Functional test

| Function Names | Testing results |
| --- | --- |
| buyTicket() | Passed |
| passFree() | Passed |
| passFreeBatch() | Passed |
| setFounder() | Passed |

# Automated Tests

## Slither

```
Reentrancy in FarmingTicketWindow.buyTicket() (FarmingTicketWindow.sol#698-704):
        External calls:
        - IBEP20(alm).safeTransferFrom(buyer,founder,ticketPrice) (FarmingTicketWindow.sol#700)
        State variables written after the call(s):
        - hasTicket[buyer] = true (FarmingTicketWindow.sol#701)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

FarmingTicketWindow.passFreeBatch(address[]).i (FarmingTicketWindow.sol#713) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Reentrancy in FarmingTicketWindow.buyTicket() (FarmingTicketWindow.sol#698-704):
        External calls:
        - IBEP20(alm).safeTransferFrom(buyer,founder,ticketPrice) (FarmingTicketWindow.sol#700)
        - IAliumCollectible(nft).mint(buyer) (FarmingTicketWindow.sol#702)
        Event emitted after the call(s):
        - TicketBought(buyer) (FarmingTicketWindow.sol#703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (FarmingTicketWindow.sol#317-328) uses assembly
        - INLINE ASM (FarmingTicketWindow.sol#324-326)
Address._functionCallWithValue(address,bytes,uint256,string) (FarmingTicketWindow.sol#425-451) uses assembly
        - INLINE ASM (FarmingTicketWindow.sol#443-446)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Pragma version>=0.4.0 (FarmingTicketWindow.sol#1) allows old versions
Pragma version>=0.4.0 (FarmingTicketWindow.sol#102) allows old versions
Pragma version^0.6.2 (FarmingTicketWindow.sol#294) allows old versions
Pragma version^0.6.0 (FarmingTicketWindow.sol#458) allows old versions
Pragma version>=0.6.0<0.8.0 (FarmingTicketWindow.sol#561) is too complex
Pragma version>=0.6.0<0.8.0 (FarmingTicketWindow.sol#588) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (FarmingTicketWindow.sol#346-352):
        - (success) = recipient.call{value: amount}() (FarmingTicketWindow.sol#350)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (FarmingTicketWindow.sol#425-451):
        - (success,returndata) = target.call{value: weiValue}(data) (FarmingTicketWindow.sol#434)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter FarmingTicketWindow.passFree(address)._account (FarmingTicketWindow.sol#706) is not in mixedCase
Parameter FarmingTicketWindow.passFreeBatch(address[])._accounts (FarmingTicketWindow.sol#711) is not in mixedCase
Parameter FarmingTicketWindow.setFounder(address)._founder (FarmingTicketWindow.sol#721) is not in mixedCase
Constant FarmingTicketWindow.ticketPrice (FarmingTicketWindow.sol#674) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (FarmingTicketWindow.sol#579)" inContext (FarmingTicketWindow.sol#573-582)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (FarmingTicketWindow.sol#638-641)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (FarmingTicketWindow.sol#647-651)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
./FarmingTicketWindow.sol analyzed (8 contracts with 75 detectors), 40 result(s) found
```

**Results**

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

## B. Contract – MasterChef

### High severity issues

No issues were found.

### Medium severity issues

#### B.1 Centralization Risks

**Description**

The role owner has the authority to :

- addPool with any lpToken to the contract
- update the reward allocation at any time
- change the setting for strong holder pool lock

**Remediation**

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

- Time-lock with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

**Status:** Fixed

**Comments from Auditee:** “Founders have a Multisig with Timelock solution in place.”

## B.2 Fewer rewards for users if SHP is disabled

| Line | Code |
| --- | --- |
| 340-356 | if (pending > 0) {<br>uint256 toTokenLock;<br>if (pool.tokenlockShare > 0) {<br>toTokenLock = pending.mul(pool.tokenlockShare).div(100);<br>//_safeAlmTransfer(msg.sender, toTokenLock);<br>// participate if reward 100K+ ALM wei<br>if (shpStatus) {<br>if (toTokenLock >= 100_000) {<br>IStrongHolder(shp).lock(msg.sender, toTokenLock);<br>} else {<br>toTokenLock = 0;<br>}<br>}<br>} else {}<br><br>_safeAlmTransfer(msg.sender, pending.sub(toTokenLock));<br>} |

**Description**
The SHP contract locks the tokens for the user if the toTokenLock calculated is more than 100,000 and shpStatus == true. If the toTokenLock < 100,000 then all the rewards are transferred to the user without any deductions.
But if shpStatus is set to false, then toTokenLock amount is still subtracted from the amount transferred to the user, but it's not locked in the Strong Holder Pool.

**Remediation**
We recommend setting toTokenLock to 0, if the shpStatus == false.

**Status:** Fixed
In version 2, the team fixed the issue with the recommended changes.

## B.3 Fewer rewards for users in staking

| Line | Code |
| --- | --- |
| 297-317 | function updatePool(uint256 _pid) public {<br>PoolInfo storage pool = poolInfo[_pid];<br>if (block.number <= pool.lastRewardBlock) {<br>return;<br>}<br><br>uint256 lpSupply = pool.lpToken.balanceOf(address(this));<br>if (lpSupply == 0) {<br>pool.lastRewardBlock = block.number;<br>return;<br>}<br><br>uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);<br><br>uint256 almReward = multiplier.mul(blockReward()).mul(pool.allocPoint).div(totalAllocPoint);<br>uint256 devReward = almReward.div(100).mul(10);<br>IAliumCashbox(cashbox).withdraw(almReward + devReward);<br>_safeAlmTransfer(devaddr, devReward);<br>pool.accALMPerShare = pool.accALMPerShare.add(almReward.mul(1e12).div(lpSupply));<br>pool.lastRewardBlock = block.number;<br>} |

**Description**

We found that staking functionality has been disabled on deployment. But it can be turned on by using the setPool() function and passing in allocPoint value > 0.

In updatePool(), the total amount of token staked for a specific pool is identified using the balance of that token in the MasterChef contract and is stored in the lpSupply variable. As the reward token and staking token are both the same,i.e. ALM, the amount stored in lpSupply will be incorrectly inflated due to the reward minted to the MasterChef contract, leading to the reward miscalculation.

This issue has a direct impact on the users because the miscalculation will lower the benefits that the users should receive, which will be continually reduced based on the rewards that have not been claimed from MasterChef by other users.

**Remediation**
The issue can be mitigated in different ways. Developers could fix this by separating the contract responsible for collecting the reward from the reward calculation contract, or not using the staking functionality and keeping it disabled.

**Status:** Fixed
In version 2, staking feature was removed and also a check for reward token != lp token was added.

## Low severity issues

### B.4 massUpdatePools() may run out-of-gas if too many tokens are added

| Line | Code |
|---|---|
| 288-293 | function massUpdatePools() public {<br>uint256 length = poolInfo.length;<br>for (uint256 pid = 0; pid < length; ++pid) {<br>updatePool(pid);<br>}<br>} |

**Description**
massUpdatePools() function loops on the poolInfo array. If too many pools are added to the contract, then it might run out of gas, because looping over all the pools and calling the updatePool() function will use a lot of gas for execution. This may have implications on calling functions add() and set()

**Remediation**
Keep a check on the number of pools added.

**Status:** Acknowledged by the Auditee
**Comments from Auditee:** “We are planning to add no more than 10-20 pools.”

## B.5 Pool with lpToken as ALM can be added again

**Description**
In the constructor, a pool with lpToken as ALM is added. But _addedLP mapping is not updated accordingly. So the owner can again add a pool with the lpToken as ALM.

**Remediation**
Update the _addedLP mapping for the ALM token in the constructor after adding the pool.

**Status:** Fixed
In version 2, a pool with lpToken == ALM cannot be added.

## B.6 blockReward() may run out-of-gas

| Line | Code |
| --- | --- |
| 275-285 | function blockReward() public view returns (uint256 reward) {<br>uint l = _blockRewards.length;<br>for (uint i = 0; i < l; i++) {<br>if (<br>block.number >= _blockRewards[i].start &&<br>block.number < _blockRewards[i].end<br>) {<br>reward = _blockRewards[i].reward;<br>}<br>}<br>} |

**Description**
blockReward() function loops on the _blockRewards array. If too many elements are added to it, then it might run out of gas, because looping over all the _blockRewards elements will use a lot of gas.

**Remediation**
We recommend adding a break statement inside the if condition, which can help save execution costs. Also, keep a check on the number of _rewards passed into the constructor.

**Status:** Fixed
In version 2, max limit for rewards was set to 10.

## Informational issues

### B.7 Missing Events for Significant Transactions

**Description**
The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions calling the following variables:

- BONUS_MULTIPLIER
- shpStatus
- devaddr
- shp

**Remediation**
We recommend emitting an event to log the update of the important variables.

**Status:** Fixed
In version 2, appropriate events were emitted.

### B.8 emergencyWithdraw() does not follow the checks-effects-interactions

| Line | Code |
| --- | --- |
| 173-180 | function emergencyWithdraw(uint256 _pid) external {<br>PoolInfo storage pool = poolInfo[_pid];<br>UserInfo storage user = userInfo[_pid][msg.sender];<br>pool.lpToken.safeTransfer(address(msg.sender), user.amount);<br>emit EmergencyWithdraw(msg.sender, _pid, user.amount);<br>user.amount = 0;<br>user.rewardDebt = 0;<br>} |

**Description**
The values for user.amount and user.rewardDebt are not zeroed until after the external call to pool.lpToken.safeTransfer(). Although the LP tokens are assumed trusted as of now, if a compromised token is added, reentrancy may allow the theft of tokens.

**Remediation**
We recommend following the checks-effects-interactions pattern. Zero out the user's state variables prior to the external safeTransfer() call (ensuring the correct amount is still passed in safeTransfer() using a local variable).

**Status:** Fixed
In version 2, the team fixed the issue with the recommended changes.

## B.9 Precision loss for dev fee

| Line | Code |
| --- | --- |
| 312 | uint256 devReward = almReward.div(100).mul(10); |

**Description**
This calculation can result in a precision loss of one decimal.
For e.g. -
almReward = 4999
devReward = (4999/100)* 10 = 490

But the calculation of 10% of almReward should result in 499. This is because of division before multiplication which is never recommended.

**Remediation**
Directly use div(10) only on almReward.

**Status:** Fixed
In version 2, the team fixed the issue by using division after multiplication.

## B.10 Incorrect Error Message

| Line | Code |
| --- | --- |
| 150 | function deposit(uint256 _pid, uint256 _amount) external canDeposit {<br>require (_pid != 0, "MasterChef: withdraw ALM by unstaking");<br><br>_deposit(_pid, _amount);<br>} |

**Description**
The error message in require (_pid != 0, "MasterChef: withdraw ALM by unstaking");  describe the error incorrectly.

**Remediation**
We recommend changing it to something appropriate.

**Status:** Fixed
As staking was removed in version 2, the require statement check was no longer needed.

# Functional test

| Function Names | Testing results |
|---|---|
| deposit() | Passed |
| withdraw() | Passed |
| stake() | Passed |
| unstake() | Passed |
| emergencyWithdraw() | Passed |
| updateMultiplier() | Passed |
| setShpStatus() | Passed |
| addPool() | Passed |
| setPool() | Passed |
| setDev() | Passed |
| setSHP() | Passed |
| poolLength() | Passed |
| pendingAlium() | Passed |
| blockReward() | Passed |
| massUpdatePools() | Passed |
| updatePool() | Passed |
| getMultiplier() | Passed |
| transferAliumOwnership() | Passed |

# Automated Tests

## Slither

```
MasterChef._safeAlmTransfer(address,uint256) (MasterChefFLAT.sol#1336-1343) ignores return value by alm.transfer(_to,ALMBal) (MasterChe
fFLAT.sol#1339)
MasterChef._safeAlmTransfer(address,uint256) (MasterChefFLAT.sol#1336-1343) ignores return value by alm.transfer(_to,_amount) (MasterCh
efFLAT.sol#1341)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

MasterChef.pendingAlium(uint256,address) (MasterChefFLAT.sol#1273-1284) performs a multiplication on the result of a division:
        -almReward = multiplier.mul(blockReward()).mul(pool.allocPoint).div(totalAllocPoint) (MasterChefFLAT.sol#1280)
        -accALMPerShare = accALMPerShare.add(almReward.mul(1e12).div(lpSupply)) (MasterChefFLAT.sol#1281)
MasterChef.updatePool(uint256) (MasterChefFLAT.sol#1308-1328) performs a multiplication on the result of a division:
        -devReward = almReward.div(100).mul(10) (MasterChefFLAT.sol#1323)
MasterChef.updatePool(uint256) (MasterChefFLAT.sol#1308-1328) performs a multiplication on the result of a division:
        -almReward = multiplier.mul(blockReward()).mul(pool.allocPoint).div(totalAllocPoint) (MasterChefFLAT.sol#1322)
        -pool.accALMPerShare = pool.accALMPerShare.add(almReward.mul(1e12).div(lpSupply)) (MasterChefFLAT.sol#1326)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

MasterChef.updatePool(uint256) (MasterChefFLAT.sol#1308-1328) uses a dangerous strict equality:
        - lpSupply == 0 (MasterChefFLAT.sol#1315)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in MasterChef._deposit(uint256,uint256) (MasterChefFLAT.sol#1345-1380):
        External calls:
        - updatePool(_pid) (MasterChefFLAT.sol#1348)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)

                 - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
         - IStrongHolder(shp).lock(msg.sender,toTokenLock) (MasterChefFLAT.sol#1359)
        - _safeAlmTransfer(msg.sender,pending.sub(toTokenLock)) (MasterChefFLAT.sol#1366)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),toService) (MasterChefFLAT.sol#1372)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChefFLAT.sol#1375)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount) (MasterChefFLAT.sol#1376)
        - user.rewardDebt = user.amount.mul(pool.accALMPerShare).div(1e12) (MasterChefFLAT.sol#1378)
Reentrancy in MasterChef._withdraw(uint256,uint256) (MasterChefFLAT.sol#1383-1415):
        External calls:
        - updatePool(_pid) (MasterChefFLAT.sol#1389)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        - IStrongHolder(shp).lock(msg.sender,toTokenLock) (MasterChefFLAT.sol#1399)
        - _safeAlmTransfer(msg.sender,pending.sub(toTokenLock)) (MasterChefFLAT.sol#1406)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (MasterChefFLAT.sol#1409)
Reentrancy in MasterChef._withdraw(uint256,uint256) (MasterChefFLAT.sol#1383-1415):
        External calls:
        - updatePool(_pid) (MasterChefFLAT.sol#1389)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
```

```
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        - IStrongHolder(shp).lock(msg.sender,toTokenLock) (MasterChefFLAT.sol#1399)
        - _safeAlmTransfer(msg.sender,pending.sub(toTokenLock)) (MasterChefFLAT.sol#1406)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChefFLAT.sol#1410)
        State variables written after the call(s):
        - user.rewardDebt = user.amount.mul(pool.accALMPerShare).div(1e12) (MasterChefFLAT.sol#1413)
Reentrancy in MasterChef.addPool(uint256,uint256,uint256,IBEP20,bool) (MasterChefFLAT.sol#1204-1233):
        External calls:
        - massUpdatePools() (MasterChefFLAT.sol#1219)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        State variables written after the call(s):
        - _addedLP[address(_lpToken)] = true (MasterChefFLAT.sol#1232)
        - poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_tokenLockShare,_depositFee)) (MasterChefFLAT.sol#1224-1231)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChefFLAT.sol#1223)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChefFLAT.sol#1184-1191):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),user.amount) (MasterChefFLAT.sol#1187)
        State variables written after the call(s):
        - user.amount = 0 (MasterChefFLAT.sol#1189)
        - user.rewardDebt = 0 (MasterChefFLAT.sol#1190)
Reentrancy in MasterChef.setPool(uint256,uint256,uint256,uint256,bool) (MasterChefFLAT.sol#1236-1252):
        External calls:

        External calls:
        - massUpdatePools() (MasterChefFLAT.sol#1242)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        State variables written after the call(s):
        - poolInfo[_pid].allocPoint = _allocPoint (MasterChefFLAT.sol#1246)
        - poolInfo[_pid].tokenlockShare = _tokenLockShare (MasterChefFLAT.sol#1247)
        - poolInfo[_pid].depositFee = _depositFee (MasterChefFLAT.sol#1248)
        - totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint) (MasterChefFLAT.sol#1250)
Reentrancy in MasterChef.setSHP(address) (MasterChefFLAT.sol#1262-1266):
        External calls:
        - IBEP20(alm).approve(shp,type()(uint256).min) (MasterChefFLAT.sol#1263)
        State variables written after the call(s):
        - shp = _shp (MasterChefFLAT.sol#1264)
Reentrancy in MasterChef.updatePool(uint256) (MasterChefFLAT.sol#1308-1328):
        External calls:
        - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        - _safeAlmTransfer(devaddr,devReward) (MasterChefFLAT.sol#1325)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        State variables written after the call(s):
        - pool.accALMPerShare = pool.accALMPerShare.add(almReward.mul(1e12).div(lpSupply)) (MasterChefFLAT.sol#1326)
        - pool.lastRewardBlock = block.number (MasterChefFLAT.sol#1327)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

MasterChef._deposit(uint256,uint256).toTokenLock (MasterChefFLAT.sol#1352) is a local variable never initialized
```

```
MasterChef._withdraw(uint256,uint256).toTokenLock (MasterChefFLAT.sol#1392) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

MasterChef.setSHP(address) (MasterChefFLAT.sol#1262-1266) ignores return value by IBEP20(alm).approve(shp,type()(uint256).min) (MasterC
hefFLAT.sol#1263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

MasterChef.updateMultiplier(uint256) (MasterChefFLAT.sol#1193-1195) should emit an event for:
        - BONUS_MULTIPLIER = multiplierNumber (MasterChefFLAT.sol#1194)
MasterChef.addPool(uint256,uint256,uint256,IBEP20,bool) (MasterChefFLAT.sol#1204-1233) should emit an event for:
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChefFLAT.sol#1223)
MasterChef.setPool(uint256,uint256,uint256,uint256,bool) (MasterChefFLAT.sol#1236-1252) should emit an event for:
        - totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint) (MasterChefFLAT.sol#1250)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in MasterChef._deposit(uint256,uint256) (MasterChefFLAT.sol#1345-1380):
        External calls:
        - updatePool(_pid) (MasterChefFLAT.sol#1348)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        - IStrongHolder(shp).lock(msg.sender,toTokenLock) (MasterChefFLAT.sol#1359)
        - _safeAlmTransfer(msg.sender,pending.sub(toTokenLock)) (MasterChefFLAT.sol#1366)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),toService) (MasterChefFLAT.sol#1372)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChefFLAT.sol#1375)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (MasterChefFLAT.sol#1379)
Reentrancy in MasterChef._withdraw(uint256,uint256) (MasterChefFLAT.sol#1383-1415):
        External calls:
        - updatePool(_pid) (MasterChefFLAT.sol#1389)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
                - IAliumCashbox(cashbox).withdraw(almReward + devReward) (MasterChefFLAT.sol#1324)
        - IStrongHolder(shp).lock(msg.sender,toTokenLock) (MasterChefFLAT.sol#1399)
        - _safeAlmTransfer(msg.sender,pending.sub(toTokenLock)) (MasterChefFLAT.sol#1406)
                - alm.transfer(_to,ALMBal) (MasterChefFLAT.sol#1339)
                - alm.transfer(_to,_amount) (MasterChefFLAT.sol#1341)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChefFLAT.sol#1410)
        Event emitted after the call(s):

        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (MasterChefFLAT.sol#1414)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChefFLAT.sol#1184-1191):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),user.amount) (MasterChefFLAT.sol#1187)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,user.amount) (MasterChefFLAT.sol#1188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (MasterChefFLAT.sol#606-617) uses assembly
        - INLINE ASM (MasterChefFLAT.sol#613-615)
Address._functionCallWithValue(address,bytes,uint256,string) (MasterChefFLAT.sol#714-740) uses assembly
        - INLINE ASM (MasterChefFLAT.sol#732-735)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MasterChefFLAT.sol#928-931)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MasterChefFLAT.sol#937-939)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
./MasterChefFLAT.sol analyzed (12 contracts with 75 detectors), 79 result(s) found
```

**Results**

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

# Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Some issues were discovered during the initial audit. In the end, the majority of the issues were totally fixed by the Auditee.

# Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the **Alium** platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Alium** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

# Audit Report
# October, 2021

For

alium
finance

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com