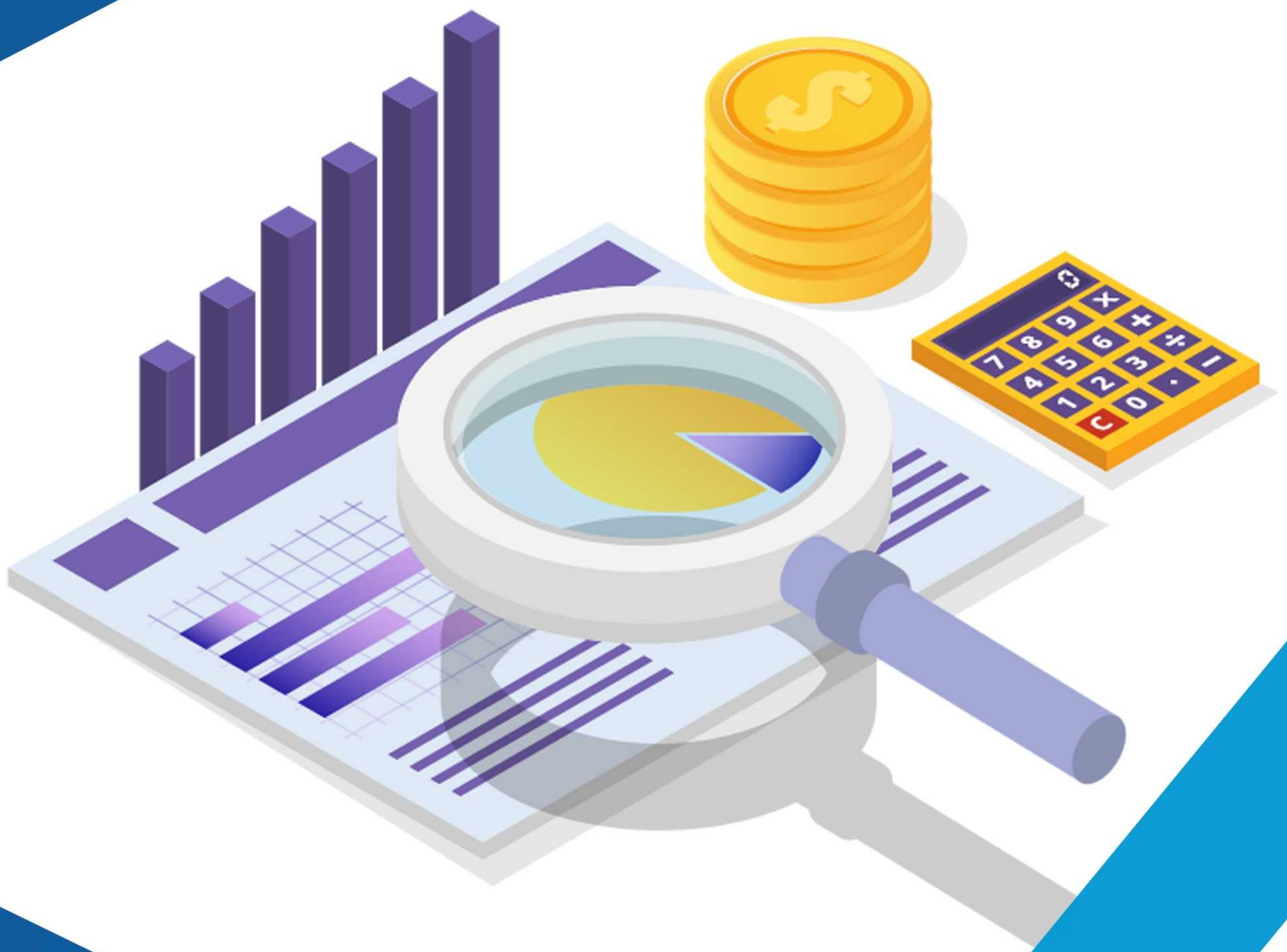
QuillAudits



Audit Report
August, 2021

MultiPad

Contents

Scope of Audit

The scope of this audit was to analyze and document the MultiPad 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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

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

High severity issues

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 to be fixed before moving to a live environment.

Medium level severity issues

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 level severity issues

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 four issues which 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	1	1	0
Closed	2	3	7	0

Introduction

During the period of **August 11, 2021 to August 16, 2021** - QuillAudits Team performed a security audit for MultiPad smart contracts.

The code for the audit was taken from the following official repo of MultiPad:
https://github.com/multipad123/mpad_quillhash

Note	Date	Commit Hash
Version 1	August	c0e3d34af0b8a6ee47a4df379ddf7c07308a1f83
Version 2	August	28797353987fce65f8d1112ecd888192bda1a115

Issues Found – Code Review / Manual Testing

High severity issues

1. Locked Tokens

Line	Code

255	function claimPurchasedTokens(
	uint256 _lockNumber
	) external validateClaim(msg.sender,_lockNumber) returns (bool)
	{
	_transfer(IDO,msg.sender,_finalSoldAmount[msg.sender].div(4));
	_recordSale[msg.sender] =
	_recordSale[msg.sender].sub(_finalSoldAmount[msg.sender].di v(4));
	reEntrance[msg.sender][_lockNumber] = true;
	_claimedByUser[msg.sender][_lockNumber] =
	_finalSoldAmount[msg.sender].div(4);
	}

Description

In the function buyToken a user can buy an amount of tokens and after he can claim them with the function claimPurchasedTokens the problem here is that these tokens are locked and cannot be claimed due to the modifier validateClaim. In fact, in this modifier, we verify if the mapping reEntrance[_userAddress] is set to false but when claiming the Tokens for the final lockNumber this value is not updated.

Remediation

The reEntrance[_userAdress] mapping need to be set to false for all the lockNumbers 1,2,3 and 4 in the final lockNumber 4 in order to give the user the ability to claim the tokens that he will buy in the future.

Status: Fixed

This issue was reported in version 1 and found fixed in version 2.

2. Race Condition

Line	Code

476	function setTokenPrice(
	uint256 _tokenPrice
	) external onlyOwner returns(bool){
	tokenPrice = _tokenPrice;
	return true;
	}
	function getTokenPrice() external view returns(uint256){
	return tokenPrice;
	}

Description

In the contract the user can call the function getTokenPrice to check the price, then he might decide to buy a number of tokens, but at the same time, the owner might have called setTokenPrice to modify the price. In the scenario where the owner's transaction gets mined first, the user could possibly buy tokens with a different price than the one that was returned by getTokenPrice.

Remediation

Add the token price as an additional parameter to the buyTokens function and add a require condition that verifies that the price provided in the parameters is equal to the current price in the smart contract.

Status: Fixed

This issue was reported in Version1 and found fixed in Version2.

Medium severity issues

3. Race Condition

Line	Code

299	function approve(
	address spender, uint256 value
	) public override returns (bool) {
	_approve(msg.sender, spender, value);
	return true;
	}

Description

The standard ERC20 implementation contains a widely known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation

Avoid using the approve function to overwrite the allowance value, use instead the increaseAllowance and decreaseAllowance functions.

Status: Acknowledged by the Auditee

The MultiPad team has acknowledged that the allowance modifications shoud be done using the increaseAllowance and decreaseAllowance functions.

4. For Loop Over Dynamic Array

Line	Code

414	function whitelistUserAdress(
	address[] calldata _userAddresses
	) external onlyOwner returns(bool){
	uint256 count = _userAddresses.length;
	for (uint256 i = 0; i < count; i++){
	_whitelistedUserAddresses.push(_userAddresses[i]);
	_whitelistedAddress[_userAddresses[i]] = true;
	}
	return true;
	}

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack.

Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status: Fixed

The MultiPad team has solved the issue by limiting the array size to 200.

5. Condition Duplication for Claiming Tokens

Line	Code

555	modifier validateClaim(
	address _userAddress, uint256 _lockNumber
	){
	require(_recordSale[_userAddress] > 0, "Not sufficient purchase Balance");
	require(_lockNumber == 1 || _lockNumber == 2 || _lockNumber == 3 || _lockNumber == 4 || _lockNumber == 5, "Invalid Lock Number");
	if(_lockNumber == 1){
	require(block.timestamp > saleEndTime + 1209600 && reEntrance[_userAddress][_lockNumber] != true, "Insufficient Unlocked Tokens");
	}
	if(_lockNumber == 2){
	require(block.timestamp > saleEndTime + 1209600 && reEntrance[_userAddress][_lockNumber] != true , "Insufficient Unlocked Tokens");
	}

Description

In the contract the user cannot claim all his purchased tokens instantly, he must wait for a specific duration to claim the first quarter (lockNumber=1) then the same goes for the 2nd, 3rd and the 4th quarter. In the validateClaim function there is a duplicated condition that might change the logic of the smart contract, that causes the user to claim the first 2 quarters in the same time.

Remediation

Change the conditions in order to give the user the ability to claim only one quarter at a time.

Status: Fixed

The MultiPad team has solved the issue by modifying the conditions to the right form.

6. Division before multiplication

Line	Code

547	function setIDOavailable(
	uint256 _IDOHardCap
	) external onlyOwner returns(bool){
	IDOAvailable = _IDOHardCap;
	return true;
	}

Description

There is no verification over the _IDOHardCap parameter, this parameter needs to be lower than the _totalSupply in order to avoid having unsynchronized data in the smart contract.

Remediation

Add a require in order to verify if the _ IDOHardCap is lower than the totalSupply.

Status: Fixed

The MultiPad Team has solved the issue by verifying that the IDOHardCap is lower than the balance of IDO.

Low level severity issues

7. Public Function That Can Be Declared External

Description

The following public functions that are never called by the contract should be declared external to save gas:

- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf(address)
- transfer(address,uint256)
- approve(address,uint256)
- allowance(address,address)
- transferFrom(address,address,uint256)

- airdropByOwner(address[], uint256[])
- increaseAllowance(address, uint256)
- decreaseAllowance (address, uint256)
- getowner()
- burn(uint256)
- transferOwnership(address)
- getWhitelistUserAdress()
- checkTokensExpected(uint256)
- getUserTokensBy(address)
- checkContractBNBBalance()
- getSoldStatus()
- getAmountPurchased(address)
- checkContractTime()
- getClaimDates()
- getClaimedTokensHistory(address)
- getBnbPricePerToken()

Remediation
Use the external attribute for functions that are not called from the contract.

Status: Fixed
The MultiPad Team has fixed the issue by declaring the public function that are not called in the smart contract as external.

8. Floating Pragma

pragma solidity ^0.6.12;

Description
The contract makes use of the floating-point pragma 0.6.12. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Fixed
The MultiPad team has solved the issue by locking the pragma version to 0.6.12.

9. Missing Address Verification

Line	Code

400	function transferOwnership(
	address newOwner
	) public onlyOwner returns (bool) {
	  _owner = newOwner;
	  return true;
	  }

426	function whitelistUserAdress(
	address[] calldata _userAddresses
	) external onlyOwner returns(bool){
	  uint256 count = _userAddresses.length;
	  for (uint256 i = 0; i < count; i++){
	  _whitelistedUserAddresses.push(_userAddresses[i]);
	  _whitelistedAddress[_userAddresses[i]] = true;
	  }
	  return true;
	  }

Description
Certain functions lack a safety check in the address, for example the address-type argument should include a zero-address test in the transferOwnership function, which is also the case for the whitelistUserAdress function.

Remediation
Add a require statement to verify that addresses passed in the parameters are different than the zero-address.

Status: Fixed
This issue was reported in version 1 and found fixed in version 2.

10. Missing Time and Amount verifications

Line	Code

437	function setSaleParameter(
	uint256 _startTime,
	uint256 _endTime,
	uint256 _minimumAmount,
	uint256 _maximumAmount,
	bool _whitelistFlag
	) external onlyOwner returns(bool){
	saleStartTime = _startTime;
	saleEndTime = _endTime;
	saleMinimumAmount = _minimumAmount;
	saleMaximumAmount = _maximumAmount;
	saleId = saleId + 1;
	whitelistFlag = _whitelistFlag;
	return true;
	}

Description

The setSaleParameter function is missing some verifications that could cause some bugs. First, a verification should be done to make sure that the _startTime is lower than the _endTime, also that the minimumAmount is lower than the maximumAmount.

Remediation

Add a require that shoud make sure that startTime is lower than the endTime and that minimumAmount is lower than the maximumAmount.

Status: Fixed

The MultiPad team has solved the issue by adding a verification to the argument's values.

11. Integer Overflow

Line	Code

270	for (uint256 i = 0; i < count; i++){
	_transfer(msg.sender, _addresses[i], _amount[i]);
	airdropcount = airdropcount + 1;
	}

448	saleMinimumAmount = _minimumAmount;
	saleMaximumAmount = _maximumAmount;
	saleId = saleId + 1;
	whitelistFlag = _whitelistFlag;

Line	Code
502	require(_contributionBNB[_userAddress] + _value >= saleMinimumAmount, "Total amount should be more than minimum limit"); require(_contributionBNB[_userAddress] + _value <= saleMaximumAmount, "Total amount should be less than maximum limit");

528	_finalSoldAmount[_userAddress] = earnedTokens;
	_contributionBNB[_userAddress] = _contributionBNB[_userAddress] + msg.value;
	IDOAvailable = IDOAvailable.sub(earnedTokens);

Description

In multiple functions in the contract, there are incrementations that were done using the usual addition operator, which might cause integer overflows.

Remediation

Use the SafeMath library to perform the mathematical operation in order to avoid integer overflows and underflows.

Status: Fixed

The MultiPad team has solved the issue by utilizing the SafeMath library to perform mathematical operations.

12. Missing Time and Amount verifications

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged by the Auditee

The MultiPad Team has accepted the risk.

13. Invalide Value in the lockNumbers conditions

Line	Code

563	modifier validateClaim(
	address _userAddress, uint256 _lockNumber
	){
	require(_recordSale[_userAddress] > 0, "Not sufficient purchase Balance");
	require(_lockNumber == 1 || _lockNumber == 2 || _lockNumber == 3 || _lockNumber == 4 || _lockNumber == 5, "Invalid Lock Number");
	if(_lockNumber == 1){
	require(block.timestamp > saleEndTime + 1209600 && reEntrance[_userAddress][_lockNumber] != true, "Insufficient Unlocked Tokens");

Description

A require condition in the validateClaim modifier contains a lockNumber equals to 5 which is not treated in the rest of the code.

Remediation

Remove the lockNumber == 5 since the logic of the contract needs just the lockNumber to be 1,2,3 or 4.

Status: Fixed

The MultiPad team fixed the issue by removing the condition from the require. Commit: e681c3aae64f2e35293da11ce0d8b07d06bcefd7

14. Unreachable if conditions

Line	Code

697	modifier checkLockingRoles(address _add, uint256 _amountRequested){
	require(_add == Team1 || _add == Team2 || _add == Partners || _add == Liquidity , "Only for Special Addresses");
	require(_amountRequested != 0, "amount should be greater than 0");
	if(_add == Reserve){
	require(block.timestamp > deploymentTime+31556926, "Tokens are locked for 1 years from TGE");
	}
	if(_add == Marketing){
	if(block.timestamp > deploymentTime && block.timestamp < deploymentTime + 2629743){
	require(specialAddBal[_add].sub(_amountRequested) >= 180000000000000000, "Amount exceded lock 1");
	}

Description
The case where _add equals Reserve or Marketing is unreachable because these two values are not included in the first require in the modifier.

Remediation
Add _add == Reserve || _add == Marketing in the require statement.

Status: Fixed
The MultiPad has fixed the issue by adding the missing conditions.

Automated Testing

Slither

```
INFO:Detectors:
MultiPad.withdrawSpecialLocked(address,uint256) (mpad.sol#605-608) ignores return value by specialAddBal[msg.sender].sub(_amount) (mpad.sol#607)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
MultiPad.constructor(string,string,uint256,address).name (mpad.sol#147) shadows:
        - MultiPad.name() (mpad.sol#188-190) (function)
MultiPad.constructor(string,string,uint256,address).symbol (mpad.sol#147) shadows:
        - MultiPad.symbol() (mpad.sol#195-197) (function)
MultiPad.constructor(string,string,uint256,address).totalSupply (mpad.sol#147) shadows:
        - MultiPad.totalSupply() (mpad.sol#202-204) (function)
        - IBEP20.totalSupply() (mpad.sol#25) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool) (mpad.sol#439-455) should emit an event for:
        - saleStartTime = _startTime (mpad.sol#447)
        - saleEndTime = _endTime (mpad.sol#448)
        - saleMinimumAmount = _minimumAmount (mpad.sol#450)
        - saleMaximumAmount = _maximumAmount (mpad.sol#451)
MultiPad.setTokenPrice(uint256) (mpad.sol#480-485) should emit an event for:
        - tokenPrice = _tokenPrice (mpad.sol#483)
MultiPad.setIDOavailable(uint256) (mpad.sol#551-554) should emit an event for:
        - IDOAvailable = _IDOHardCap (mpad.sol#552)
MultiPad.setBnbPricePerToken(uint256) (mpad.sol#688-691) should emit an event for:
        - pricePerToken = _price (mpad.sol#689)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
MultiPad.constructor(string,string,uint256,address).owner (mpad.sol#147) lacks a zero-check on :
                - _owner = owner (mpad.sol#152)
MultiPad.transferOwnership(address).newOwner (mpad.sol#402) lacks a zero-check on :
                - _owner = newOwner (mpad.sol#403)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
MultiPad.getUserTokensBy(address) (mpad.sol#621-640) compares to a boolean constant:
        -reEntrance[_userAddress][1] == true (mpad.sol#627)
MultiPad.getUserTokensBy(address) (mpad.sol#621-640) compares to a boolean constant:
        -reEntrance[_userAddress][3] == true (mpad.sol#633)
MultiPad.getUserTokensBy(address) (mpad.sol#621-640) compares to a boolean constant:
        -reEntrance[_userAddress][4] == true (mpad.sol#636)
MultiPad.getUserTokensBy(address) (mpad.sol#621-640) compares to a boolean constant:
        -reEntrance[_userAddress][2] == true (mpad.sol#630)
MultiPad.checkSaleValidations(address,uint256) (mpad.sol#500-510) compares to a boolean constant:
        -require(bool,string)(_whitelistedAddress[_userAddress] == true,Address not Whitelisted) (mpad.sol#502)
MultiPad.checkSaleValidations(address,uint256) (mpad.sol#500-510) compares to a boolean constant:
        -whitelistFlag == true (mpad.sol#501)
MultiPad.validateClaim(address,uint256) (mpad.sol#567-584) compares to a boolean constant:
        -require(bool,string)(block.timestamp > saleEndTime + 2419200 && reEntrance[_userAddress][_lockNumber] != true,Insufficient Unlocked Tokens) (mpad.sol#578)
MultiPad.validateClaim(address,uint256) (mpad.sol#567-584) compares to a boolean constant:
        -require(bool,string)(block.timestamp > saleEndTime + 3628800 && reEntrance[_userAddress][_lockNumber] != true,Insufficient Unlocked Tokens) (mpad.sol#581)
MultiPad.validateClaim(address,uint256) (mpad.sol#567-584) compares to a boolean constant:
        -require(bool,string)(block.timestamp > saleEndTime + 1209600 && reEntrance[_userAddress][_lockNumber] != true,Insufficient Unlocked Tokens) (mpad.sol#575)
MultiPad.validateClaim(address,uint256) (mpad.sol#567-584) compares to a boolean constant:
        -require(bool,string)(block.timestamp > saleEndTime + 1209600 && reEntrance[_userAddress][_lockNumber] != true,Insufficient Unlocked Tokens) (mpad.sol#572)
MultiPad.checkLockedAddresses(address) (mpad.sol#597-600) compares to a boolean constant:
        -require(bool,string)(_addressLocked[_lockedAddresses] != true,Locking Address) (mpad.sol#598)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
SafeMath.mod(uint256,uint256) (mpad.sol#90-93) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Parameter MultiPad.airdropByOwner(address[],uint256[])._addresses (mpad.sol#266) is not in mixedCase
Parameter MultiPad.airdropByOwner(address[],uint256[])._amount (mpad.sol#266) is not in mixedCase
Parameter MultiPad.whitelistUserAdress(address[])._userAddresses (mpad.sol#416) is not in mixedCase
Parameter MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool)._startTime (mpad.sol#440) is not in mixedCase
```

```
MultiPad.Team2 (mpad.sol#138) should be constant
MultiPad._decimals (mpad.sol#104) should be constant
MultiPad.decimalBalancer (mpad.sol#131) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
name() should be declared external:
        - MultiPad.name() (mpad.sol#188-190)
symbol() should be declared external:
        - MultiPad.symbol() (mpad.sol#195-197)
totalSupply() should be declared external:
        - MultiPad.totalSupply() (mpad.sol#202-204)
decimals() should be declared external:
        - MultiPad.decimals() (mpad.sol#209-211)
balanceOf(address) should be declared external:
        - MultiPad.balanceOf(address) (mpad.sol#218-220)
allowance(address,address) should be declared external:
        - MultiPad.allowance(address,address) (mpad.sol#228-230)
transfer(address,uint256) should be declared external:
        - MultiPad.transfer(address,uint256) (mpad.sol#242-245)
transferFrom(address,address,uint256) should be declared external:
        - MultiPad.transferFrom(address,address,uint256) (mpad.sol#255-259)
airdropByOwner(address[],uint256[]) should be declared external:
        - MultiPad.airdropByOwner(address[],uint256[]) (mpad.sol#266-277)
approve(address,uint256) should be declared external:
        - MultiPad.approve(address,uint256) (mpad.sol#301-304)
increaseAllowance(address,uint256) should be declared external:
        - MultiPad.increaseAllowance(address,uint256) (mpad.sol#329-332)
decreaseAllowance(address,uint256) should be declared external:
        - MultiPad.decreaseAllowance(address,uint256) (mpad.sol#344-347)
burn(uint256) should be declared external:
        - MultiPad.burn(uint256) (mpad.sol#366-368)
getowner() should be declared external:
        - MultiPad.getowner() (mpad.sol#379-381)
transferOwnership(address) should be declared external:
        - MultiPad.transferOwnership(address) (mpad.sol#402-405)
getWhitelistUserAdress() should be declared external:
        - MultiPad.getWhitelistUserAdress() (mpad.sol#428-430)
checkTokensExpected(uint256) should be declared external:
        - MultiPad.checkTokensExpected(uint256) (mpad.sol#513-515)
getUserTokensBy(address) should be declared external:
        - MultiPad.getUserTokensBy(address) (mpad.sol#621-640)
checkContractBNBBalance() should be declared external:
        - MultiPad.checkContractBNBBalance() (mpad.sol#645-647)
getSoldStatus() should be declared external:
        - MultiPad.getSoldStatus() (mpad.sol#650-653)
getAmountPurchased(address) should be declared external:
        - MultiPad.getAmountPurchased(address) (mpad.sol#655-658)
checkContractTime() should be declared external:
        - MultiPad.checkContractTime() (mpad.sol#661-663)
getClaimDates() should be declared external:
        - MultiPad.getClaimDates() (mpad.sol#666-672)
getClaimedTokensHistory(address) should be declared external:
        - MultiPad.getClaimedTokensHistory(address) (mpad.sol#677-683)
getBnbPricePerToken() should be declared external:
        - MultiPad.getBnbPricePerToken() (mpad.sol#696-698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:mpad.sol analyzed (3 contracts with 75 detectors), 101 result(s) found
```

```
Parameter MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool)._endTime (mpad.sol#441) is not in mixedCase
Parameter MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool)._minimumAmount (mpad.sol#442) is not in mixedCase
Parameter MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool)._maximumAmount (mpad.sol#443) is not in mixedCase
Parameter MultiPad.setSaleParameter(uint256,uint256,uint256,uint256,bool)._whitelistFlag (mpad.sol#444) is not in mixedCase
Parameter MultiPad.setTokenPrice(uint256)._tokenPrice (mpad.sol#481) is not in mixedCase
Parameter MultiPad.checkTokensExpected(uint256)._value (mpad.sol#513) is not in mixedCase
Parameter MultiPad.setIDOavailable(uint256)._IDOHardCap (mpad.sol#551) is not in mixedCase
Parameter MultiPad.claimPurchasedTokens(uint256)._lockNumber (mpad.sol#559) is not in mixedCase
Parameter MultiPad.checkWhitelistedAddress(address)._userAddress (mpad.sol#589) is not in mixedCase
Parameter MultiPad.withdrawSpecialLocked(address,uint256)._toAddress (mpad.sol#605) is not in mixedCase
Parameter MultiPad.withdrawSpecialLocked(address,uint256)._amount (mpad.sol#605) is not in mixedCase
Parameter MultiPad.withdrawBNB(uint256)._amount (mpad.sol#613) is not in mixedCase
Parameter MultiPad.getUserTokensBy(address)._userAddress (mpad.sol#621) is not in mixedCase
Parameter MultiPad.getAmountPurchased(address)._userAddress (mpad.sol#655) is not in mixedCase
Parameter MultiPad.getClaimedTokensHistory(address)._userAddress (mpad.sol#677) is not in mixedCase
Parameter MultiPad.setBnbPricePerToken(uint256)._price (mpad.sol#688) is not in mixedCase
Variable MultiPad._whitelistedAddress (mpad.sol#112) is not in mixedCase
Variable MultiPad._lockingTimeForSale (mpad.sol#113) is not in mixedCase
Variable MultiPad._recordSale (mpad.sol#114) is not in mixedCase
Variable MultiPad._addressLocked (mpad.sol#115) is not in mixedCase
Variable MultiPad._finalSoldAmount (mpad.sol#116) is not in mixedCase
Variable MultiPad._contributionBNB (mpad.sol#119) is not in mixedCase
Variable MultiPad._claimedByUser (mpad.sol#120) is not in mixedCase
Variable MultiPad.IDOAvailable (mpad.sol#132) is not in mixedCase
Variable MultiPad.Reserve (mpad.sol#135) is not in mixedCase
Variable MultiPad.Marketing (mpad.sol#136) is not in mixedCase
Variable MultiPad.Teaml (mpad.sol#137) is not in mixedCase
Variable MultiPad.Team2 (mpad.sol#138) is not in mixedCase
Variable MultiPad.Liquidity (mpad.sol#139) is not in mixedCase
Variable MultiPad.IDO (mpad.sol#140) is not in mixedCase
Variable MultiPad.Partners (mpad.sol#141) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Reserve] = 25000000 * (10 ** uint256(_decimals)) (mpad.sol#163)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Marketing] = 20000000 * (10 ** uint256(_decimals)) (mpad.sol#164)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Teaml] = 18000000 * (10 ** uint256(_decimals)) (mpad.sol#165)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Team2] = 2000000 * (10 ** uint256(_decimals)) (mpad.sol#166)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Liquidity] = 10000000 * (10 ** uint256(_decimals)) (mpad.sol#167)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[IDO] = 15000000 * (10 ** uint256(_decimals)) (mpad.sol#168)
MultiPad.initiateValues() (mpad.sol#162-177) uses literals with too many digits:
        - specialAddBal[Partners] = 10000000 * (10 ** uint256(_decimals)) (mpad.sol#169)
MultiPad.getSoldStatus() (mpad.sol#650-653) uses literals with too many digits:
        - _totalAvailable = 150000000000000000000000000 (mpad.sol#651)
MultiPad.slitherConstructorVariables() (mpad.sol#96-824) uses literals with too many digits:
        - decimalBalancer = 1000000000 (mpad.sol#131)
MultiPad.slitherConstructorVariables() (mpad.sol#96-824) uses literals with too many digits:
        - IDOAvailable = 150000000000000000000000000 (mpad.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
MultiPad.IDO (mpad.sol#140) should be constant
MultiPad.Liquidity (mpad.sol#139) should be constant
MultiPad.Marketing (mpad.sol#136) should be constant
MultiPad.Partners (mpad.sol#141) should be constant
MultiPad.Reserve (mpad.sol#135) should be constant
MultiPad.Teaml (mpad.sol#137) should be constant
MultiPad.Team2 (mpad.sol#138) should be constant
MultiPad._decimals (mpad.sol#104) should be constant
```

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.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the MultiPad Contract. 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 MultiPad Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are decently written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, The contract initially had some vulnerabilities of different levels, however, the majority of these vulnerabilities have been corrected and fixed by the MultiPad team and now the code is more robust and secure.

MultiPad

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com