MythX

REPORT 6068B2C3259F900012B5F587

Created	Sat Apr 03 2021 18:24:03 GMT+0000 (Coordinated Universal Time)
Number of analyses	1

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
4cd7f7fa-e0a3-47a9-bf99-1df3ee79c717	/contracts/masterchefv2.sol	35

Analysis 4cd7f7fa-e0a3-47a9-bf99-1df3ee79c717

MythX

Started	Sat Apr 03 2021 18:24:05 GMT+0000 (Coordinated Universal Time)
Finished	Sat Apr 03 2021 18:39:52 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Masterchefv2.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	12	23

ISSUES

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "add" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/masterchefv2.sol

Locations

```
// Add a new lp to the pool. Can only be called by the owner.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner nonDuplicated(_lpToken) {
require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
if (_withUpdate) {
massUpdatePools();
}
uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
totalAllocPoint = totalAllocPoint.add(_allocPoint);
poolExistence[_lpToken] = true;
poolInfo.push(PoolInfo({
lpToken : _lpToken,
allocPoint : _allocPoint,
lastRewardBlock : lastRewardBlock,
accRewardPerShare : 0,
depositFeeBP : _depositFeeBP
}));
}

// Update the given pool's REWARD allocation point and deposit fee. Can only be called by the owner.
function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
if (_withUpdate) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "set" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`/contracts/masterchefv2.sol`

Locations

```
function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
if (_withUpdate) {
massUpdatePools();
}
totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
poolInfo[_pid].allocPoint = _allocPoint;
poolInfo[_pid].depositFeeBP = _depositFeeBP;
}

// Return reward multiplier over the given _from to _to block.
function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
return _to.sub(_from).mul(getBonusMultiplier());
}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "compoundAll" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`/contracts/masterchefv2.sol`

Locations

```
}

function compound(uint256 _pid) public {
updatePool(0);
_compound(_pid);
}

function _compound(uint256 _pid) internal bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "compound" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/masterchefv2.sol

Locations

```

function _compound(uint256 _pid) internal bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];

if (user.amount > 0) {
updatePool(_pid);
uint256 pending = user.amount.mul(pool.accRewardPerShare).div(1e12).sub(user.rewardDebt);
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "deposit" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/masterchefv2.sol

Locations

```
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
updatePool(_pid);
if (user.amount > 0) {
uint256 pending = user.amount.mul(pool.accRewardPerShare).div(1e12).sub(user.rewardDebt);
if (pending > 0) {
safeRewardTransfer(msg.sender, pending);
}
}
if (_amount > 0) {
pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
if (pool.depositFeeBP > 0) {
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
pool.lpToken.safeTransfer(feeAddress, depositFee);
user.amount = user.amount.add(_amount).sub(depositFee);
} else {
user.amount = user.amount.add(_amount);
}
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
require(user.amount >= _amount, "withdraw: not good");
updatePool(_pid);
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "withdraw" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/masterchefv2.sol

Locations

```
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
require(user.amount >= _amount, "withdraw: not good");
updatePool(_pid);
uint256 pending = user.amount.mul(pool.accRewardPerShare).div(1e12).sub(user.rewardDebt);
if (pending > 0) {
safeRewardTransfer(msg.sender, pending);
}
if (_amount > 0) {
user.amount = user.amount.sub(_amount);
pool.lpToken.safeTransfer(address(msg.sender), _amount);
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Withdraw(msg.sender, _pid, _amount);
}

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
uint256 amount = user.amount;
user.amount = 0;
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "emergencyWithdraw" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/masterchefv2.sol

Locations

```
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
uint256 amount = user.amount;
user.amount = 0;
user.rewardDebt = 0;
pool.lpToken.safeTransfer(address(msg.sender), amount);
emit EmergencyWithdraw(msg.sender, _pid, amount);
}

// Safe reward transfer function, just in case if rounding error causes pool to not have enough REWARDs.
function safeRewardTransfer(address _to, uint256 _amount) internal {
uint256 rewardBal = reward.balanceOf(address(this));
bool transferSuccess = false;
if (_amount > rewardBal) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "dev" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`/contracts/masterchefv2.sol`

Locations

```
}

function setFeeAddress(address _feeAddress) public {
require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
feeAddress = _feeAddress;
emit SetFeeAddress(msg.sender, _feeAddress);
}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setFeeAddress" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`/contracts/masterchefv2.sol`

Locations

```
require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
feeAddress = _feeAddress;
emit SetFeeAddress(msg.sender, _feeAddress);
}

//Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
function updateEmissionRate(uint256 _rewardPerBlock) public onlyOwner {
massUpdatePools();
rewardPerBlock = _rewardPerBlock;
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "massUpdatePools" in contract "MasterChefV2" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`/contracts/masterchefv2.sol`

Locations

```
// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public {
PoolInfo storage pool = poolInfo[_pid];
if (block.number <= pool.lastRewardBlock) {
return;
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "compoundAll" in contract "MasterChefV2" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`/contracts/masterchefv2.sol`

Locations

```
}

function _compound(uint256 _pid) internal bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`/contracts/masterchefv2.sol`

Locations

```
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
pool.lpToken.safeTransfer(feeAddress, depositFee);
user.amount = user.amount.add(_amount).sub(depositFee);
} else {
user.amount = user.amount.add(_amount);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`/contracts/masterchefv2.sol`

Locations

```
}

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`/contracts/masterchefv2.sol`

Locations

```
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`/contracts/masterchefv2.sol`

Locations

```

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`/contracts/masterchefv2.sol`

Locations

```

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant bonusCheck {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
require(user.amount >= _amount, "withdraw: not good");
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```
user.amount = user.amount.add(_amount).sub(depositFee);
} else {
user.amount = user.amount.add(_amount);
}
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```
user.amount = user.amount.add(_amount);
}
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```
user.amount = user.amount.add(_amount).sub(depositFee);
} else {
user.amount = user.amount.add(_amount);
}
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```
}
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```
}
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterchefv2.sol

Locations

```

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
uint256 amount = user.amount;
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
poolExistence[_lpToken] = true;
poolInfo.push(PoolInfo({
lpToken : _lpToken,
allocPoint : _allocPoint,
lastRewardBlock : lastRewardBlock,
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
poolInfo.push(PoolInfo({
lpToken : _lpToken,
allocPoint : _allocPoint,
lastRewardBlock : lastRewardBlock,
accRewardPerShare : 0,
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
if (block.number > pool.lastRewardBlock && lpSupply != 0) {
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 rewardReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accRewardPerShare = accRewardPerShare.add(rewardReward.mul(1e12).div(lpSupply));
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 rewardReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accRewardPerShare = accRewardPerShare.add(rewardReward.mul(1e12).div(lpSupply));
}
return user.amount.mul(accRewardPerShare).div(1e12).sub(user.rewardDebt);
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (lpSupply == 0 || pool.allocPoint == 0) {
pool.lastRewardBlock = block.number;
return;
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
}
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 rewardReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
if (rewardReward > 0) {
reward.mint(devaddr, rewardReward.div(10));
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
uint256 rewardReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
if (rewardReward > 0) {
reward.mint(devaddr, rewardReward.div(10));
reward.mint(address(this), rewardReward);
pool.accRewardPerShare = pool.accRewardPerShare.add(rewardReward.mul(1e12).div(lpSupply));
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`/contracts/masterchefv2.sol`

Locations

```
uint256 length = poolInfo.length;
for (uint256 pid = 0; pid < length; ++pid) {
_compound(pid);
}
}
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`/contracts/masterchefv2.sol`

Locations

```
if (block.number > pool.lastRewardBlock && lpSupply != 0) {
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 rewardReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accRewardPerShare = accRewardPerShare.add(rewardReward.mul(1e12).div(lpSupply));
}
```