MythX

REPORT 60F85298C1C74200189763B0

Created	Wed Jul 21 2021 17:00:08 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	60b6a744a6e1845c77c6e3dc

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
7ae72c68-2fce-466c-8cf8-155c6ac51ecb	/contracts-v1/masterchef.sol	74

Analysis 7ae72c68-2fce-466c-8cf8-155c6ac51ecb MythX

Started	Wed Jul 21 2021 17:00:11 GMT+0000 (Coordinated Universal Time)
Finished	Wed Jul 21 2021 17:16:28 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts-V1/Masterchef.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	34	40

ISSUES

MEDIUM Function could be marked as external.

SWC-000

The function definition of "renounceOwnership" 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-v1/masterchef.sol

Locations

```
359 * thereby removing any functionality that is only available to the owner.
360 */
361 function renounceOwnership() public virtual onlyOwner {
362 emit OwnershipTransferred(_owner, address(0));
363 _owner = address(0);
364 }
365
366 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "transferOwnership" 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-v1/masterchef.sol

Locations

```
368 * Can only be called by the current owner.
369 */
370 function transferOwnership(address newOwner) public virtual onlyOwner {
371 require(newOwner != address(0), "Ownable: new owner is the zero address");
372 emit OwnershipTransferred(_owner, newOwner);
373 _owner = newOwner;
374 }
375 }
376
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "decimals" 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-v1/masterchef.sol

Locations

```
1073 * @dev Returns the token decimals.
1074 */
1075 function decimals() public override view returns (uint8) {
1076 return _decimals;
1077 }
1078
1079 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "symbol" 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-v1/masterchef.sol

Locations

```
1080 * @dev Returns the token symbol.
1081 */
1082 function symbol() public override view returns (string memory) {
1083 return _symbol;
1084 }
1085
1086 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transfer" 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-v1/masterchef.sol

Locations

```
1106 * - the caller must have a balance of at least `amount`.
1107 */
1108 function transfer(address recipient, uint256 amount) public override returns (bool) {
1109 _transfer(_msgSender(), recipient, amount);
1110 return true;
1111 }
1112
1113 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "allowance" 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-v1/masterchef.sol

Locations

```
1114 * @dev See {BEP20-allowance}.
1115 */
1116 function allowance(address owner, address spender) public override view returns (uint256) {
1117 return _allowances[owner][spender];
1118 }
1119
1120 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "approve" 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-v1/masterchef.sol

Locations

```
1125 * - `spender` cannot be the zero address.
1126 */
1127 function approve(address spender, uint256 amount) public override returns (bool) {
1128 _approve(_msgSender(), spender, amount);
1129 return true;
1130 }
1131
1132 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferFrom" 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-v1/masterchef.sol

Locations

```
1142 * `amount`.
1143 */
1144 function transferFrom(
1145 address sender,
1146 address recipient,
1147 uint256 amount
1148 ) public override returns (bool) {
1149 _transfer(sender, recipient, amount);
1150 _approve(
1151 sender,
1152 _msgSender(),
1153 _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance")
1154 );
1155 return true;
1156 }
1157
1158 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "increaseAllowance" 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-v1/masterchef.sol

Locations

```
1168 * - `spender` cannot be the zero address.
1169 */
1170 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
1171 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
1172 return true;
1173 }
1174
1175 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decreaseAllowance" 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-v1/masterchef.sol

Locations

```
1187 * `subtractedValue`.
1188 */
1189 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
1190 _approve(
1191 _msgSender(),
1192 spender,
1193 _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zero")
1194 );
1195 return true;
1196 }
1197
1198 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" 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-v1/masterchef.sol

Locations

```
1204 * - `msg.sender` must be the token owner
1205 */
1206 function mint(uint256 amount) public onlyOwner returns (bool) {
1207 _mint(_msgSender(), amount);
1208 return true;
1209 }
1210
1211 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" 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-v1/masterchef.sol

Locations

```
1400
1401 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
1402 function mint(address _to, uint256 _amount) public onlyOwner {
1403 _mint(_to, _amount);
1404 _moveDelegates(address(0), _delegates[_to], _amount);
1405 }
1406
1407 /// @dev overrides transfer function to meet tokenomics of TFLASH
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "isExcludedFromAntiWhale" 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-v1/masterchef.sol

Locations

```
1517 * @dev Returns the address is excluded from antiWhale or not.
1518 */
1519 function isExcludedFromAntiWhale(address _account) public view returns (bool) {
1520 return _excludedFromAntiWhale[_account];
1521 }
1522
1523 // To receive BNB from flashSwapRouter when swapping
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateTransferTaxRate" 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-v1/masterchef.sol

Locations

```
1528 * Can only be called by the current operator.
1529 */
1530 function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
1531 require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "TFLASH::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
1532 emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
1533 transferTaxRate = _transferTaxRate;
1534 }
1535
1536 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateBurnRate" 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-v1/masterchef.sol

Locations

```
1538 * Can only be called by the current operator.
1539 */
1540 function updateBurnRate(uint16 _burnRate) public onlyOperator {
1541 require(_burnRate <= 100, "TFLASH::updateBurnRate: Burn rate must not exceed the maximum rate.");
1542 emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
1543 burnRate = _burnRate;
1544 }
1545
1546 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateMaxTransferAmountRate" 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-v1/masterchef.sol

Locations

```
1548 * Can only be called by the current operator.
1549 */
1550 function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
1551 require(_maxTransferAmountRate <= 10000, "TFLASH::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
1552 emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
1553 maxTransferAmountRate = _maxTransferAmountRate;
1554 }
1555
1556 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateMinAmountToLiquify" 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-v1/masterchef.sol

Locations

```
1558 * Can only be called by the current operator.
1559 */
1560 function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
1561 emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
1562 minAmountToLiquify = _minAmount;
1563 }
1564
1565 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setExcludedFromAntiWhale" 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-v1/masterchef.sol

Locations

```
1567 * Can only be called by the current operator.
1568 */
1569 function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
1570 _excludedFromAntiWhale[_account] = _excluded;
1571 }
1572
1573 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateSwapAndLiquifyEnabled" 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-v1/masterchef.sol

Locations

```
1575 * Can only be called by the current operator.
1576 */
1577 function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
1578 emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
1579 swapAndLiquifyEnabled = _enabled;
1580 }
1581
1582 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateflashSwapRouter" 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-v1/masterchef.sol

Locations

```
1584 * Can only be called by the current operator.
1585 */
1586 function updateflashSwapRouter(address _router) public onlyOperator {
1587 flashSwapRouter = IUniswapV2Router02(_router);
1588 flashSwapPair = IUniswapV2Factory(flashSwapRouter.factory()).getPair(address(this), flashSwapRouter.WETH());
1589 require(flashSwapPair != address(0), "TFLASH::updateflashSwapRouter: Invalid pair address.");
1590 emit flashSwapRouterUpdated(msg.sender, address(flashSwapRouter), flashSwapPair);
1591 }
1592
1593 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "transferOperator" 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-v1/masterchef.sol

Locations

```
1602 * Can only be called by the current operator.
1603 */
1604 function transferOperator(address newOperator) public onlyOperator {
1605 require(newOperator != address(0), "TFLASH::transferOperator: new operator is the zero address");
1606 emit OperatorTransferred(_operator, newOperator);
1607 _operator = newOperator;
1608 }
1609
1610 // Copied and modified from YAM code:
```

MEDIUM

Function could be marked as external.

SWC-000

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-v1/masterchef.sol

Locations

```
1950 // Add a new lp to the pool. Can only be called by the owner.
1951 // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
1952 function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
1953 require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
1954 require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add: invalid harvest interval");
1955 if (_withUpdate) {
1956 massUpdatePools();
1957 }
1958 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1959 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1960 poolInfo.push(PoolInfo({
1961 lpToken: _lpToken,
1962 allocPoint: _allocPoint,
1963 lastRewardBlock: lastRewardBlock,
1964 accFlashPerShare: 0,
1965 depositFeeBP: _depositFeeBP,
1966 harvestInterval: _harvestInterval
1967 }));
1968 }
1969
1970
```

MEDIUM

Function could be marked as external.

SWC-000

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-v1/masterchef.sol

Locations

```
1970
1971 // Update the given pool's FLASH allocation point and deposit fee. Can only be called by the owner.
1972 function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
1973 require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
1974 require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "set: invalid harvest interval");
1975 if (_withUpdate) {
1976 massUpdatePools();
1977 }
1978 totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
1979 poolInfo[_pid].allocPoint = _allocPoint;
1980 poolInfo[_pid].depositFeeBP = _depositFeeBP;
1981 poolInfo[_pid].harvestInterval = _harvestInterval;
1982 }
1983
1984 // Return reward multiplier over the given _from to _to block.
```

MEDIUM Function could be marked as external.

SWC-000 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-v1/masterchef.sol

Locations

```
2039
2040 // Deposit LP tokens to MasterChef for FLASH allocation.
2041 function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
2042 PoolInfo storage pool = poolInfo[_pid];
2043 UserInfo storage user = userInfo[_pid][msg.sender];
2044 updatePool(_pid);
2045 if (_amount > 0 && address(flashReferral) != address(0) && _referrer != address(0) && _referrer != msg.sender) {
2046 flashReferral.recordReferral(msg.sender, _referrer);
2047 }
2048 payOrLockupPendingFlash(_pid);
2049 if (_amount > 0) {
2050 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2051 if (address(pool.lpToken) == address(flash)) {
2052 uint256 transferTax = _amount.mul(flash.transferTaxRate()).div(10000);
2053 _amount = _amount.sub(transferTax);
2054 }
2055 if (pool.depositFeeBP > 0) {
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
2061 user.amount = user.amount.add(_amount);
2062 }
2063 }
2064 user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2065 emit Deposit(msg.sender, _pid, _amount);
2066 }
2067
2068 // Withdraw LP tokens from MasterChef.
```

MEDIUM

Function could be marked as external.

SWC-000

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-v1/masterchef.sol

Locations

```
2067
2068 // Withdraw LP tokens from MasterChef.
2069 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
2070 PoolInfo storage pool = poolInfo[_pid];
2071 UserInfo storage user = userInfo[_pid][msg.sender];
2072 require(user.amount >= _amount, "withdraw: not good");
2073 updatePool(_pid);
2074 payOrLockupPendingFlash(_pid);
2075 if (_amount > 0) {
2076 user.amount = user.amount.sub(_amount);
2077 pool.lpToken.safeTransfer(address(msg.sender), _amount);
2078 }
2079 user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2080 emit Withdraw(msg.sender, _pid, _amount);
2081 }
2082
2083 // Withdraw without caring about rewards. EMERGENCY ONLY.
```

MEDIUM

Function could be marked as external.

SWC-000

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-v1/masterchef.sol

Locations

```
2082
2083 // Withdraw without caring about rewards. EMERGENCY ONLY.
2084 function emergencyWithdraw(uint256 _pid) public nonReentrant {
2085 PoolInfo storage pool = poolInfo[_pid];
2086 UserInfo storage user = userInfo[_pid][msg.sender];
2087 uint256 amount = user.amount;
2088 user.amount = 0;
2089 user.rewardDebt = 0;
2090 user.rewardLockedUp = 0;
2091 user.nextHarvestUntil = 0;
2092 pool.lpToken.safeTransfer(address(msg.sender), amount);
2093 emit EmergencyWithdraw(msg.sender, _pid, amount);
2094 }
2095
2096 // Pay or lockup pending FLASHs.
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setDevAddress" 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-v1/masterchef.sol

Locations

```
2135
2136 // Update dev address by the previous dev.
2137 function setDevAddress(address _devAddress) public {
2138 require(msg.sender == devAddress, "setDevAddress: FORBIDDEN");
2139 require(_devAddress != address(0), "setDevAddress: ZERO");
2140 devAddress = _devAddress;
2141 }
2142
2143 // Update marketing address by the previous marketing address.
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setMarketingAddress" 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-v1/masterchef.sol

Locations

```
2142
2143 // Update marketing address by the previous marketing address.
2144 function setMarketingAddress(address _marketingAddress) public {
2145 require(msg.sender == marketingAddress, "setMarketingAddress: FORBIDDEN");
2146 require(_marketingAddress != address(0), "setMarketingAddress: ZERO");
2147 marketingAddress = _marketingAddress;
2148 }
2149
2150 function setFeeAddrBaMa(address _feeAddBb) public {
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setFeeAddrBaMa" 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-v1/masterchef.sol

Locations

```
2148 }
2149
2150 function setFeeAddrBaMa(address _feeAddBb) public {
2151 require(msg.sender == feeAddBb, "setFeeAddrBaMa: FORBIDDEN");
2152 require(_feeAddBb != address(0), "setFeeAddrBaMa: ZERO");
2153 feeAddBb = _feeAddBb;
2154 }
2155
2156 function setFeeAddrStMa(address _feeAddSt) public {
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setFeeAddrStMa" 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-v1/masterchef.sol

Locations

```
2154 }
2155
2156 function setFeeAddrStMa(address _feeAddSt) public {
2157 require(msg.sender == feeAddSt, "setFeeAddrStMa: FORBIDDEN");
2158 require(_feeAddSt != address(0), "setFeeAddrStMa: ZERO");
2159 feeAddSt = _feeAddSt;
2160 }
2161
2162 // Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateEmissionRate" 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-v1/masterchef.sol

Locations

```
2161
2162 // Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
2163 function updateEmissionRate(uint256 _flashPerBlock) public onlyOwner {
2164 emit EmissionRateUpdated(msg.sender, flashPerBlock, _flashPerBlock);
2165 flashPerBlock = _flashPerBlock;
2166 }
2167
2168 // Update the flash referral contract address by the owner
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setFlashReferral" 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-v1/masterchef.sol

Locations

```
2167
2168 // Update the flash referral contract address by the owner
2169 function setFlashReferral(IFlashReferral _flashReferral) public onlyOwner {
2170 flashReferral = _flashReferral;
2171 }
2172
2173 // Update referral commission rate by the owner
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setReferralCommissionRate" 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-v1/masterchef.sol

Locations

```
2172
2173 // Update referral commission rate by the owner
2174 function setReferralCommissionRate(uint16 _referralCommissionRate) public onlyOwner {
2175 require(_referralCommissionRate <= MAXIMUM_REFERRAL_COMMISSION_RATE, "setReferralCommissionRate: invalid referral commission rate basis points");
2176 referralCommissionRate = _referralCommissionRate;
2177 }
2178
2179 // Pay referral commission to the referrer who referred this user.
```

MEDIUM

Multiple calls are executed in the same transaction.

SWC-113

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts-v1/masterchef.sol

Locations

```
514
515 // solhint-disable-next-line avoid-low-level-calls
516 (bool success, bytes memory returndata) = target.call{ value: value }(data);
517 return _verifyCallResult(success, returndata, errorMessage);
518 }
```

LOW

A floating pragma is set.

SWC-103

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
3 // File: @uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol
4
5 pragma solidity ^0.5.0;
6
7 interface IUniswapV2Factory {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
23  // File: @uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol
24
25  pragma solidity >=0.5.0;
26
27  interface IUniswapV2Pair {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
78  // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol
79
80  pragma solidity >=0.6.2;
81
82  interface IUniswapV2Router01 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
176 // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol
177
178 pragma solidity >=0.6.2;
179
180 interface IUniswapV2Router02 is IUniswapV2Router01 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
221 // File: @openzeppelin/contracts/utils/ReentrancyGuard.sol
222
223 pragma solidity >=0.6.0 <0.8.0;
224
225 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
284 // File: @openzeppelin/contracts/utils/Context.sol
285
286 pragma solidity >=0.6.0 <0.8.0;
287
288 /*
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
309 // File: @openzeppelin/contracts/access/Ownable.sol
310
311 pragma solidity >=0.6.0 <0.8.0;
312
313 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is """>=0.6.2<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
398 // File: @openzeppelin/contracts/utils/Address.sol
399
400 pragma solidity >=0.6.2 <0.8.0;
401
402 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
588 // File: contracts/libs/SafeBEP20.sol
589
590 pragma solidity ^0.6.0;
591
592 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is """>=0.4.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
686 // File: contracts/libs/IBEP20.sol
687
688 pragma solidity >=0.4.0;
689
690 interface IBEP20 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
785 // File: @openzeppelin/contracts/math/SafeMath.sol
786
787 pragma solidity >=0.6.0 <0.8.0;
788
789 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is """>=0.4.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts-v1/masterchef.sol

Locations

```
1001 // File: contracts/libs/BEP20.sol
1002
1003 pragma solidity >=0.4.0;
1004
1005 /**
```

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-v1/masterchef.sol

Locations

```
2049 if (_amount > 0) {
2050 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2051 if (address(pool.lpToken) == address(flash)) {
2052 uint256 transferTax = _amount.mul(flash.transferTaxRate()).div(10000);
2053 _amount = _amount.sub(transferTax);
```

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-v1/masterchef.sol

Locations

```
2049 if (_amount > 0) {
2050 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2051 if (address(pool.lpToken) == address(flash)) {
2052 uint256 transferTax = _amount.mul(flash.transferTaxRate()).div(10000);
2053 _amount = _amount.sub(transferTax);
```

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-v1/masterchef.sol

Locations

```
2053 _amount = _amount.sub(transferTax);
2054 }
2055 if (pool.depositFeeBP > 0) {
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
```

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-v1/masterchef.sol

Locations

```
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
2061 user.amount = user.amount.add(_amount);
2062 }
2063 }
```

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-v1/masterchef.sol

Locations

```
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
2061 user.amount = user.amount.add(_amount);
2062 }
2063 }
```

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-v1/masterchef.sol

Locations

```
2062 }
2063 }
2064 user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2065 emit Deposit(msg.sender, _pid, _amount);
2066 }
```

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-v1/masterchef.sol

Locations

```
2062 }
2063 }
2064 user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2065 emit Deposit(msg.sender, _pid, _amount);
2066 }
```

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-v1/masterchef.sol

Locations

```
2062 }
2063 }
2064 user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2065 emit Deposit(msg.sender, _pid, _amount);
2066 }
```

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-v1/masterchef.sol

Locations

```
278 // By storing the original value once again, a refund is triggered (see
279 // https://eips.ethereum.org/EIPS/eip-2200)
280 _status = _NOT_ENTERED;
281 }
282 }
```

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-v1/masterchef.sol

Locations

```
2054 }
2055 if (pool.depositFeeBP > 0) {
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
```

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-v1/masterchef.sol

Locations

```
2055 if (pool.depositFeeBP > 0) {
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
```

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-v1/masterchef.sol

Locations

```
2055 if (pool.depositFeeBP > 0) {
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
```

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-v1/masterchef.sol

Locations

```
510 */
511 function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
512 require(address(this).balance >= value, "Address: insufficient balance for call");
513 require(isContract(target), "Address: call to non-contract");
514
```

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-v1/masterchef.sol

Locations

```
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
```

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-v1/masterchef.sol

Locations

```
2056 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
```

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-v1/masterchef.sol

Locations

```
2057 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 } else {
2061 user.amount = user.amount.add(_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-v1/masterchef.sol

Locations

```
2057 | pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058 | pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059 | user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060 | } else {
2061 | user.amount = user.amount.add(_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-v1/masterchef.sol

Locations

```
1742 | returns (uint256)
1743 | {
1744 | require(blockNumber < block.number, "TFLASH::getPriorVotes: not yet determined");
1745 |
1746 | uint32 nCheckpoints = numCheckpoints[account];
```

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-v1/masterchef.sol

Locations

```
1815 | internal
1816 | {
1817 | uint32 blockNumber = safe32(block.number, "TFLASH::_writeCheckpoint: block number exceeds 32 bits");
1818 |
1819 | if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
```

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-v1/masterchef.sol

Locations

```
1956  massUpdatePools();
1957  }
1958  uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1959  totalAllocPoint = totalAllocPoint.add(_allocPoint);
1960  poolInfo.push(PoolInfo({
```

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-v1/masterchef.sol

Locations

```
1956  massUpdatePools();
1957  }
1958  uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1959  totalAllocPoint = totalAllocPoint.add(_allocPoint);
1960  poolInfo.push(PoolInfo({
```

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-v1/masterchef.sol

Locations

```
1993  uint256 accFlashPerShare = pool.accFlashPerShare;
1994  uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1995  if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1996  uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1997  uint256 flashReward = multiplier.mul(flashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
```

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-v1/masterchef.sol

Locations

```
1994 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1995 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1996 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1997 uint256 flashReward = multiplier.mul(flashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1998 accFlashPerShare = accFlashPerShare.add(flashReward.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-v1/masterchef.sol

Locations

```
2019 function updatePool(uint256 _pid) public {
2020 PoolInfo storage pool = poolInfo[_pid];
2021 if (block.number <= pool.lastRewardBlock) {
2022 return;
2023 }
```

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-v1/masterchef.sol

Locations

```
2024 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2025 if (lpSupply == 0 || pool.allocPoint == 0) {
2026 pool.lastRewardBlock = block.number;
2027 return;
2028 }
```

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-v1/masterchef.sol

Locations

```
2029
2030  // Mint
2031  uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
2032  uint256 flashReward = multiplier.mul(flashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
2033  flash.mint(marketingAddress, flashReward.div(30));
```

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-v1/masterchef.sol

Locations

```
2035  flash.mint(address(this), flashReward);
2036  pool.accFlashPerShare = pool.accFlashPerShare.add(flashReward.mul(1e12).div(lpSupply));
2037  pool.lastRewardBlock = block.number;
2038  }
2039
```

LOW

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).

SWC-123

Source file

/contracts-v1/masterchef.sol

Locations

```
2022 return;
2023 }
2024 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2025 if (lpSupply == 0 || pool.allocPoint == 0) {
2026 pool.lastRewardBlock = block.number;
```

Source file

/contracts-v1/masterchef.sol

Locations

```
1850 //
1851 // Have fun reading it. Hopefully it's bug-free. God bless.
1852 contract MasterChef is Ownable, ReentrancyGuard {
1853 using SafeMath for uint256;
1854 using SafeBEP20 for IBEP20;
1855
1856 // Info of each user.
1857 struct UserInfo {
1858 uint256 amount; // How many LP tokens the user has provided.
1859 uint256 rewardDebt; // Reward debt. See explanation below.
1860 uint256 rewardLockedUp; // Reward locked up.
1861 uint256 nextHarvestUntil; // When can the user harvest again.
1862 //
1863 // We do some fancy math here. Basically, any point in time, the amount of FLASHs
1864 // entitled to a user but is pending to be distributed is:
1865 //
1866 // pending reward = (user.amount * pool.accFlashPerShare) - user.rewardDebt
1867 //
1868 // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
1869 // 1. The pool's `accFlashPerShare` (and `lastRewardBlock`) gets updated.
1870 // 2. User receives the pending reward sent to his/her address.
1871 // 3. User's `amount` gets updated.
1872 // 4. User's `rewardDebt` gets updated.
1873 }
1874
1875 // Info of each pool.
1876 struct PoolInfo {
1877 IBEP20 lpToken; // Address of LP token contract.
1878 uint256 allocPoint; // How many allocation points assigned to this pool. FLASHs to distribute per block.
1879 uint256 lastRewardBlock; // Last block number that FLASHs distribution occurs.
1880 uint256 accFlashPerShare; // Accumulated FLASHs per share, times 1e12. See below.
1881 uint16 depositFeeBP; // Deposit fee in basis points
1882 uint256 harvestInterval; // Harvest interval in seconds
1883 }
1884
1885 // The FLASH TOKEN!
1886 TheFlashToken public flash;
1887 // Dev address.
1888 address public devAddress;
1889 // Marketing address.
1890 address public marketingAddress;
1891 // Deposit Fee address
1892 address public feeAddBb;
1893 address public feeAddSt;
1894 // FLASH tokens created per block.
```

```
uint256 public flashPerBlock;
// Bonus muliplier for early flash makers.
uint256 public constant BONUS_MULTIPLIER = 1;
// Max harvest interval: 14 days.
uint256 public constant MAXIMUM_HARVEST_INTERVAL = 14 days;

// Info of each pool.
PoolInfo[] public poolInfo;
// Info of each user that stakes LP tokens.
mapping(uint256 => mapping(address => UserInfo)) public userInfo;
// Total allocation points. Must be the sum of all allocation points in all pools.
uint256 public totalAllocPoint = 0;
// The block number when FLASH mining starts.
uint256 public startBlock;
// Total locked up rewards
uint256 public totalLockedUpRewards;

// Flash referral contract address.
IFlashReferral public flashReferral;
// Referral commission rate in basis points.
uint16 public referralCommissionRate = 300;
// Max referral commission rate: 10%.
uint16 public constant MAXIMUM_REFERRAL_COMMISSION_RATE = 1000;

event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmissionRateUpdated(address indexed caller, uint256 previousAmount, uint256 newAmount);
event ReferralCommissionPaid(address indexed user, address indexed referrer, uint256 commissionAmount);
event RewardLockedUp(address indexed user, uint256 indexed pid, uint256 amountLockedUp);

constructor(
TheFlashToken _flash,
uint256 _startBlock,
uint256 _flashPerBlock,
address _devAddress,
address _marketingAddress,
address _feeAddBb,
address _feeAddSt
) public {
flash = _flash;
startBlock = _startBlock;
flashPerBlock = _flashPerBlock;

devAddress = _devAddress;
marketingAddress = _marketingAddress;

feeAddBb = _feeAddBb;
feeAddSt = _feeAddSt;
}

function poolLength() external view returns (uint256) {
return poolInfo.length;
}

// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add: invalid harvest interval");
if (_withUpdate) {
massUpdatePools();
}
```

```
uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
totalAllocPoint = totalAllocPoint.add(_allocPoint);
poolInfo.push(PoolInfo({
lpToken: _lpToken,
allocPoint: _allocPoint,
lastRewardBlock: lastRewardBlock,
accFlashPerShare: 0,
depositFeeBP: _depositFeeBP,
harvestInterval: _harvestInterval
}));
}



// Update the given pool's FLASH allocation point and deposit fee. Can only be called by the owner.
function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "set: invalid harvest interval");
if (_withUpdate) {
massUpdatePools();
}
totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
poolInfo[_pid].allocPoint = _allocPoint;
poolInfo[_pid].depositFeeBP = _depositFeeBP;
poolInfo[_pid].harvestInterval = _harvestInterval;
}

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

// View function to see pending FLASHs on frontend.
function pendingFlash(uint256 _pid, address _user) external view returns (uint256) {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][_user];
uint256 accFlashPerShare = pool.accFlashPerShare;
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (block.number > pool.lastRewardBlock && lpSupply != 0) {
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 flashReward = multiplier.mul(flashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accFlashPerShare = accFlashPerShare.add(flashReward.mul(1e12).div(lpSupply));
}
uint256 pending = user.amount.mul(accFlashPerShare).div(1e12).sub(user.rewardDebt);
return pending.add(user.rewardLockedUp);
}

// View function to see if user can harvest FLASHs.
function canHarvest(uint256 _pid, address _user) public view returns (bool) {
UserInfo storage user = userInfo[_pid][_user];
return block.timestamp >= user.nextHarvestUntil;
}

// Update reward variables for all pools. Be careful of gas spending!
function massUpdatePools() public {
uint256 length = poolInfo.length;
for (uint256 pid = 0; pid < length; ++pid) {
updatePool(pid);
}
}

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

```
2021  if (block.number <= pool.lastRewardBlock) {
2022  return;
2023  }
2024  uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2025  if (lpSupply == 0 || pool.allocPoint == 0) {
2026  pool.lastRewardBlock = block.number;
2027  return;
2028  }
2029
2030  // Mint
2031  uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
2032  uint256 flashReward = multiplier.mul(flashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
2033  flash.mint(marketingAddress, flashReward.div(30));
2034  flash.mint(devAddress, flashReward.mul(2).div(30));
2035  flash.mint(address(this), flashReward);
2036  pool.accFlashPerShare = pool.accFlashPerShare.add(flashReward.mul(1e12).div(lpSupply));
2037  pool.lastRewardBlock = block.number;
2038  }
2039
2040  // Deposit LP tokens to MasterChef for FLASH allocation.
2041  function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
2042  PoolInfo storage pool = poolInfo[_pid];
2043  UserInfo storage user = userInfo[_pid][msg.sender];
2044  updatePool(_pid);
2045  if (_amount > 0 && address(flashReferral) != address(0) && _referrer != address(0) && _referrer != msg.sender) {
2046  flashReferral.recordReferral(msg.sender, _referrer);
2047  }
2048  payOrLockupPendingFlash(_pid);
2049  if (_amount > 0) {
2050  pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2051  if (address(pool.lpToken) == address(flash)) {
2052  uint256 transferTax = _amount.mul(flash.transferTaxRate()).div(10000);
2053  _amount = _amount.sub(transferTax);
2054  }
2055  if (pool.depositFeeBP > 0) {
2056  uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2057  pool.lpToken.safeTransfer(feeAddBb, depositFee);
2058  pool.lpToken.safeTransfer(feeAddSt, depositFee);
2059  user.amount = user.amount.add(_amount).sub(depositFee).sub(depositFee);
2060  } else {
2061  user.amount = user.amount.add(_amount);
2062  }
2063  }
2064  user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2065  emit Deposit(msg.sender, _pid, _amount);
2066  }
2067
2068  // Withdraw LP tokens from MasterChef.
2069  function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
2070  PoolInfo storage pool = poolInfo[_pid];
2071  UserInfo storage user = userInfo[_pid][msg.sender];
2072  require(user.amount >= _amount, "withdraw: not good");
2073  updatePool(_pid);
2074  payOrLockupPendingFlash(_pid);
2075  if (_amount > 0) {
2076  user.amount = user.amount.sub(_amount);
2077  pool.lpToken.safeTransfer(address(msg.sender), _amount);
2078  }
2079  user.rewardDebt = user.amount.mul(pool.accFlashPerShare).div(1e12);
2080  emit Withdraw(msg.sender, _pid, _amount);
2081  }
2082
2083  // 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;
user.rewardDebt = 0;
user.rewardLockedUp = 0;
user.nextHarvestUntil = 0;
pool.lpToken.safeTransfer(address(msg.sender), amount);
emit EmergencyWithdraw(msg.sender, _pid, amount);
}

// Pay or lockup pending FLASHs.
function payOrLockupPendingFlash(uint256 _pid) internal {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];

if (user.nextHarvestUntil == 0) {
user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval);
}

uint256 pending = user.amount.mul(pool.accFlashPerShare).div(1e12).sub(user.rewardDebt);
if (canHarvest(_pid, msg.sender)) {
if (pending > 0 || user.rewardLockedUp > 0) {
uint256 totalRewards = pending.add(user.rewardLockedUp);

// reset lockup
totalLockedUpRewards = totalLockedUpRewards.sub(user.rewardLockedUp);
user.rewardLockedUp = 0;
user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval);

// send rewards
safeFlashTransfer(msg.sender, totalRewards);
payReferralCommission(msg.sender, totalRewards);
}
} else if (pending > 0) {
user.rewardLockedUp = user.rewardLockedUp.add(pending);
totalLockedUpRewards = totalLockedUpRewards.add(pending);
emit RewardLockedUp(msg.sender, _pid, pending);
}
}

// Safe flash transfer function, just in case if rounding error causes pool to not have enough FLASHs.
function safeFlashTransfer(address _to, uint256 _amount) internal {
uint256 flashBal = flash.balanceOf(address(this));
if (_amount > flashBal) {
flash.transfer(_to, flashBal);
} else {
flash.transfer(_to, _amount);
}
}

// Update dev address by the previous dev.
function setDevAddress(address _devAddress) public {
require(msg.sender == devAddress, "setDevAddress: FORBIDDEN");
require(_devAddress != address(0), "setDevAddress: ZERO");
devAddress = _devAddress;
}

// Update marketing address by the previous marketing address.
function setMarketingAddress(address _marketingAddress) public {
require(msg.sender == marketingAddress, "setMarketingAddress: FORBIDDEN");
require(_marketingAddress != address(0), "setMarketingAddress: ZERO");
```

```
2147    marketingAddress = _marketingAddress;
2148    }
2149
2150    function setFeeAddrBaMa(address _feeAddBb) public {
2151    require(msg.sender == feeAddBb, "setFeeAddrBaMa: FORBIDDEN");
2152    require(_feeAddBb != address(0), "setFeeAddrBaMa: ZERO");
2153    feeAddBb = _feeAddBb;
2154    }
2155
2156    function setFeeAddrStMa(address _feeAddSt) public {
2157    require(msg.sender == feeAddSt, "setFeeAddrStMa: FORBIDDEN");
2158    require(_feeAddSt != address(0), "setFeeAddrStMa: ZERO");
2159    feeAddSt = _feeAddSt;
2160    }
2161
2162    // Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
2163    function updateEmissionRate(uint256 _flashPerBlock) public onlyOwner {
2164    emit EmissionRateUpdated(msg.sender, flashPerBlock, _flashPerBlock);
2165    flashPerBlock = _flashPerBlock;
2166    }
2167
2168    // Update the flash referral contract address by the owner
2169    function setFlashReferral(IFlashReferral _flashReferral) public onlyOwner {
2170    flashReferral = _flashReferral;
2171    }
2172
2173    // Update referral commission rate by the owner
2174    function setReferralCommissionRate(uint16 _referralCommissionRate) public onlyOwner {
2175    require(_referralCommissionRate <= MAXIMUM_REFERRAL_COMMISSION_RATE, "setReferralCommissionRate: invalid referral commission rate basis points");
2176    referralCommissionRate = _referralCommissionRate;
2177    }
2178
2179    // Pay referral commission to the referrer who referred this user.
2180    function payReferralCommission(address _user, uint256 _pending) internal {
2181    if (address(flashReferral) != address(0) && referralCommissionRate > 0) {
2182    address referrer = flashReferral.getReferrer(_user);
2183    uint256 commissionAmount = _pending.mul(referralCommissionRate).div(10000);
2184
2185    if (referrer != address(0) && commissionAmount > 0) {
2186    flash.mint(referrer, commissionAmount);
2187    flashReferral.recordReferralCommission(referrer, commissionAmount);
2188    emit ReferralCommissionPaid(_user, referrer, commissionAmount);
2189    }
2190    }
2191    }
2192    }
```