MythX

REPORT 60D34E0732D2B300182BB418

Created	Wed Jun 23 2021 15:06:47 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	60d316478bfa1246dff293e4

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
6847ed12-a1d8-4ae5-b0fe-1ae964c7967b	contracts/WhaleKiller.sol	65

Analysis 6847ed12-a1d8-4ae5-b0fe-1ae964c7967b MythX

Started	Wed Jun 23 2021 15:06:49 GMT+0000 (Coordinated Universal Time)
Finished	Wed Jun 23 2021 15:24:06 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Remythx
Main Source File	Contracts/WhaleKiller.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	34	31

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/WhaleKiller.sol
Locations

```
556 * thereby removing any functionality that is only available to the owner.
557 */
558 function renounceOwnership() public virtual onlyOwner {
559 emit OwnershipTransferred(_owner, address(0));
560 _owner = address(0);
561 }
562
563 /**
```

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/WhaleKiller.sol

Locations

```
565 * Can only be called by the current owner.
566 */
567 function transferOwnership(address newOwner) public virtual onlyOwner {
568 require(newOwner != address(0), "Ownable: new owner is the zero address");
569 emit OwnershipTransferred(_owner, newOwner);
570 _owner = newOwner;
571 }
572 }
```

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/WhaleKiller.sol

Locations

```
643 * @dev Returns the token decimals.
644 */
645 function decimals() public override view returns (uint8) {
646 return _decimals;
647 }
648
649 /**
```

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/WhaleKiller.sol

Locations

```
650 * @dev Returns the token symbol.
651 */
652 function symbol() public override view returns (string memory) {
653 return _symbol;
654 }
655
656 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
676 * - the caller must have a balance of at least `amount`.
677 */
678 function transfer(address recipient, uint256 amount) public override returns (bool) {
679 _transfer(_msgSender(), recipient, amount);
680 return true;
681 }
682
683 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
684 * @dev See {BEP20-allowance}.
685 */
686 function allowance(address owner, address spender) public override view returns (uint256) {
687 return _allowances[owner][spender];
688 }
689
690 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
695 * - `spender` cannot be the zero address.
696 */
697 function approve(address spender, uint256 amount) public override returns (bool) {
698 _approve(_msgSender(), spender, amount);
699 return true;
700 }
701
702 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
712 * `amount`.
713 */
714 function transferFrom(
715 address sender,
716 address recipient,
717 uint256 amount
718 ) public override returns (bool) {
719 _transfer(sender, recipient, amount);
720 _approve(
721 sender,
722 _msgSender(),
723 _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance")
724 );
725 return true;
726 }
727
728 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
738 * - `spender` cannot be the zero address.
739 */
740 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
741 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
742 return true;
743 }
744
745 /**
```

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/WhaleKiller.sol

Locations

```
757 * `subtractedValue`.
758 */
759 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
760 _approve(
761 _msgSender(),
762 spender,
763 _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zero")
764 );
765 return true;
766 }
767
768 /**
```

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/WhaleKiller.sol

Locations

```
774 * - `msg.sender` must be the token owner
775 */
776 function mint(uint256 amount) public onlyOwner returns (bool) {
777 _mint(_msgSender(), amount);
778 return true;
779 }
780
781 /**
```

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/WhaleKiller.sol

Locations

```
1246
1247 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
1248 function mint(address _to, uint256 _amount) public onlyOwner {
1249 _mint(_to, _amount);
1250 _moveDelegates(address(0), _delegates[_to], _amount);
1251 }
1252
1253 /// @dev overrides transfer function to meet tokenomics of WHALE
```

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/WhaleKiller.sol

Locations

```
1363 * @dev Returns the address is excluded from antiWhale or not.
1364 */
1365 function isExcludedFromAntiWhale(address _account) public view returns (bool) {
1366 return _excludedFromAntiWhale[_account];
1367 }
1368
1369 // To receive BNB from whaleRouter 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/WhaleKiller.sol

Locations

```
1374 * Can only be called by the current operator.
1375 */
1376 function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
1377 require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "WHALE::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
1378 emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
1379 transferTaxRate = _transferTaxRate;
1380 }
1381
1382 /**
```

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/WhaleKiller.sol

Locations

```
1384 * Can only be called by the current operator.
1385 */
1386 function updateBurnRate(uint16 _burnRate) public onlyOperator {
1387 require(_burnRate <= 100, "WHALE::updateBurnRate: Burn rate must not exceed the maximum rate.");
1388 emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
1389 burnRate = _burnRate;
1390 }
1391
1392 /**
```

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/WhaleKiller.sol

Locations

```
1394 * Can only be called by the current operator.
1395 */
1396 function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
1397 require(_maxTransferAmountRate <= 10000, "WHALE::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
1398 emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
1399 maxTransferAmountRate = _maxTransferAmountRate;
1400 }
1401
1402 /**
```

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/WhaleKiller.sol

Locations

```
1404 * Can only be called by the current operator.
1405 */
1406 function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
1407 emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
1408 minAmountToLiquify = _minAmount;
1409 }
1410
1411 /**
```

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/WhaleKiller.sol

Locations

```
1413 * Can only be called by the current operator.
1414 */
1415 function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
1416 _excludedFromAntiWhale[_account] = _excluded;
1417 }
1418
1419 /**
```

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/WhaleKiller.sol

Locations

```
1421 * Can only be called by the current operator.
1422 */
1423 function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
1424 emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
1425 swapAndLiquifyEnabled = _enabled;
1426 }
1427
1428 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "UpdateSwapEnabled" 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/WhaleKiller.sol

Locations

```
1429 * @dev Update the swapEnabled. Can only be called by the current Owner.
1430 */
1431 function UpdateSwapEnabled(bool _enabled) public onlyOwner {
1432 emit SwapEnabledUpdated(msg.sender, _enabled);
1433 swapEnabled = _enabled;
1434 }
1435
1436 /**
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateWhaleRouter" 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/WhaleKiller.sol

Locations

```
1438 * Can only be called by the current operator.
1439 */
1440 function updateWhaleRouter(address _router) public onlyOperator {
1441 whaleRouter = IUniswapV2Router02(_router);
1442 whalePair = IUniswapV2Factory(whaleRouter.factory()).getPair(address(this), whaleRouter.WETH());
1443 require(whalePair != address(0), "WHALE::updateWhaleRouter: Invalid pair address.");
1444 emit WhaleRouterUpdated(msg.sender, address(whaleRouter), whalePair);
1445 }
1446
1447 /**
```

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/WhaleKiller.sol

Locations

```
1456 * Can only be called by the current operator.
1457 */
1458 function transferOperator(address newOperator) public onlyOperator {
1459 require(newOperator != address(0), "WHALE::transferOperator: new operator is the zero address");
1460 emit OperatorTransferred(_operator, newOperator);
1461 _operator = newOperator;
1462 }
1463
1464 // 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/WhaleKiller.sol

Locations

```
1825
1826 // Add a new lp to the pool. Can only be called by the owner.
1827 function add(
1828 uint256 _allocPoint,
1829 IBEP20 _lpToken,
1830 uint16 _depositFeeBP,
1831 bool _withUpdate
1832 ) public onlyOwner nonDuplicated(_lpToken) {
1833 require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "add: invalid deposit fee basis points");
1834 if (_withUpdate) {
1835 massUpdatePools();
1836 }
1837 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1838 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1839 poolExistence[_lpToken] = true;
1840 poolInfo.push(
1841 PoolInfo({
1842 lpToken: _lpToken,
1843 allocPoint: _allocPoint,
1844 lastRewardBlock: lastRewardBlock,
1845 accWhalePerShare: 0,
1846 depositFeeBP: _depositFeeBP
1847 })
1848 );
1849 poolIdForLpAddress[_lpToken] = poolInfo.length - 1;
1850 }
1851
1852 // Update the given pool's WHALE allocation point and deposit fee. Can only be called by the owner.
```

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/WhaleKiller.sol

Locations

```
1851
1852 // Update the given pool's WHALE allocation point and deposit fee. Can only be called by the owner.
1853 function set(
1854 uint256 _pid,
1855 uint256 _allocPoint,
1856 uint16 _depositFeeBP,
1857 bool _withUpdate
1858 ) public onlyOwner {
1859 require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "set: invalid deposit fee basis points");
1860 if (_withUpdate) {
1861 massUpdatePools();
1862 }
1863 totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
1864 _allocPoint
1865 );
1866 poolInfo[_pid].allocPoint = _allocPoint;
1867 poolInfo[_pid].depositFeeBP = _depositFeeBP;
1868 }
1869
1870 // 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/WhaleKiller.sol

Locations

```
1933
1934 // Deposit LP tokens to MasterChef for WHALE allocation.
1935 function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
1936 PoolInfo storage pool = poolInfo[_pid];
1937 UserInfo storage user = userInfo[_pid][msg.sender];
1938 updatePool(_pid);
1939 payOrLockupPendingWhale(_pid);
1940 if (_amount > 0) {
1941 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
1942 if (address(pool.lpToken) == address(whale)) {
1943 uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
1944 _amount = _amount.sub(transferTax);
1945 }
1946 if (pool.depositFeeBP > 0) {
1947 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1948 user.amount = user.amount.add(_amount).sub(depositFee);
1949 pool.lpToken.safeTransfer(feeAddress, depositFee);
1950 } else {
1951 user.amount = user.amount.add(_amount);
1952 }
1953 }
1954
1955 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
1956 emit Deposit(msg.sender, _pid, _amount);
1957 }
1958
1959 // Deposit LP tokens to MasterChef for WHALE allocation with referral.
```

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/WhaleKiller.sol

Locations

```
1958
1959 // Deposit LP tokens to MasterChef for WHALE allocation with referral.
1960 function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
1961 PoolInfo storage pool = poolInfo[_pid];
1962 UserInfo storage user = userInfo[_pid][msg.sender];
1963 updatePool(_pid);
1964 if (_amount > 0 && _referrer != address(0) && _referrer == address(_referrer) && _referrer != msg.sender) {
1965 setReferral(msg.sender, _referrer);
1966 }
1967 payOrLockupPendingWhale(_pid);
1968
1969 if (_amount > 0) {
1970 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
1971 if (address(pool.lpToken) == address(whale)) {
1972 uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
1973 _amount = _amount.sub(transferTax);
1974 }
1975 if (pool.depositFeeBP > 0) {
1976 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1977 user.amount = user.amount.add(_amount).sub(depositFee);
1978 pool.lpToken.safeTransfer(feeAddress, depositFee);
1979 } else {
1980 user.amount = user.amount.add(_amount);
1981 }
1982 }
1983
1984 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
1985 emit Deposit(msg.sender, _pid, _amount);
1986 }
1987
1988 // 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/WhaleKiller.sol

Locations

```
1987
1988 // Withdraw LP tokens from MasterChef.
1989 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
1990 PoolInfo storage pool = poolInfo[_pid];
1991 UserInfo storage user = userInfo[_pid][msg.sender];
1992 require(user.amount >= _amount, "withdraw: not good");
1993 updatePool(_pid);
1994 payOrLockupPendingWhale(_pid);
1995
1996 if (_amount > 0) {
1997 user.amount = user.amount.sub(_amount);
1998 pool.lpToken.safeTransfer(address(msg.sender), _amount);
1999 }
2000 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
2001 emit Withdraw(msg.sender, _pid, _amount);
2002 }
2003
2004 // 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/WhaleKiller.sol

Locations

```
2003
2004 // Withdraw without caring about rewards. EMERGENCY ONLY.
2005 function emergencyWithdraw(uint256 _pid) public nonReentrant {
2006 PoolInfo storage pool = poolInfo[_pid];
2007 UserInfo storage user = userInfo[_pid][msg.sender];
2008 pool.lpToken.safeTransfer(address(msg.sender), user.amount);
2009 emit EmergencyWithdraw(msg.sender, _pid, user.amount);
2010 user.amount = 0;
2011 user.rewardDebt = 0;
2012 user.rewardLockedUp = 0;
2013 }
2014
2015 // Pay or lockup pending WHALEs.
```

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/WhaleKiller.sol

Locations

```
2051
2052 // Update dev address by the previous dev.
2053 function setDevAddress(address _devaddr) public {
2054 require(_devaddr != address(0), "dev: invalid address");
2055 require(msg.sender == devAddr, "dev: wut?");
2056 devAddr = _devaddr;
2057 emit SetDevAddress(msg.sender, _devaddr);
2058 }
2059
2060 // Update fee address by the previous fee address.
```

MEDIUM

Function could be marked as external.

SWC-000

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/WhaleKiller.sol

Locations

```
2059
2060 // Update fee address by the previous fee address.
2061 function setFeeAddress(address _feeAddress) public {
2062 require(_feeAddress != address(0), "setFeeAddress: invalid address");
2063 require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
2064 feeAddress = _feeAddress;
2065 emit SetFeeAddress(msg.sender, _feeAddress);
2066 }
2067
```

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/WhaleKiller.sol

Locations

```
2068
2069 // updateEmissionRate
2070 function updateEmissionRate(uint256 _whalePerBlock) public onlyOwner {
2071 massUpdatePools();
2072 emit EmissionRateUpdated(msg.sender, whalePerBlock, _whalePerBlock);
2073 whalePerBlock = _whalePerBlock;
2074 }
2075
2076 // updateHarvestTime, how many blocks
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateHarvestTime" 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/WhaleKiller.sol

Locations

```
2075
2076 // updateHarvestTime, how many blocks
2077 function updateHarvestTime(uint256 _harvestTime) public onlyOwner {
2078 harvestTime = _harvestTime;
2079 emit UpdateHarvestTime(msg.sender, harvestTime, _harvestTime);
2080 }
2081
2082 // updateStartBlockHarvest
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateStartBlockHarvest" 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/WhaleKiller.sol

Locations

```
2081
2082 // updateStartBlockHarvest
2083 function updateStartBlockHarvest(uint256 _startBlockHarvest) public onlyOwner {
2084 startBlockHarvest = _startBlockHarvest;
2085 emit UpdateStartBlockHarvest(msg.sender, startBlockHarvest, _startBlockHarvest);
2086 }
2087
2088 // Set Referral Address for a user
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateReferralBonusBp" 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/WhaleKiller.sol

Locations

```
2113 // Initially set to 3%, this gives the ability to increase or decrease the bonus referral percentage based on
2114 // community voting and feedback.
2115 function updateReferralBonusBp(uint256 _newRefBonusBp) public onlyOwner {
2116 require(_newRefBonusBp <= MAXIMUM_REFERRAL_BP, "updateRefBonusPercent: invalid referral bonus basis points");
2117 require(_newRefBonusBp != refBonusBP, "updateRefBonusPercent: same referral bonus set");
2118 uint256 previousRefBonusBP = refBonusBP;
2119 refBonusBP = _newRefBonusBp;
2120 emit ReferralBonusBpChanged(previousRefBonusBP, _newRefBonusBp);
2121 }
2122 }
```

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/WhaleKiller.sol

Locations

```
5   // SPDX-License-Identifier: MIT
6
7   pragma solidity >=0.6.0 <0.8.0;
8
9   /**
```

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/WhaleKiller.sol

Locations

```
506 }
507
508 pragma solidity >=0.6.0 <0.8.0;
509
510 /**
```

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/WhaleKiller.sol

Locations

```
902 // File: @uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol
903
904 pragma solidity >=0.5.0;
905
906 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/WhaleKiller.sol

Locations

```
957 // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol
958
959 pragma solidity ^0.6.2;
960
961 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/WhaleKiller.sol

Locations

```
1055 // File: @uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol
1056
1057 pragma solidity ^0.6.2;
1058
```

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/WhaleKiller.sol

Locations

```
2007 UserInfo storage user = userInfo[_pid][msg.sender];
2008 pool.lpToken.safeTransfer(address(msg.sender), user.amount);
2009 emit EmergencyWithdraw(msg.sender, _pid, user.amount);
2010 user.amount = 0;
2011 user.rewardDebt = 0;
```

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/WhaleKiller.sol

Locations

```
2008  pool.lpToken.safeTransfer(address(msg.sender), user.amount);
2009  emit EmergencyWithdraw(msg.sender, _pid, user.amount);
2010  user.amount = 0;
2011  user.rewardDebt = 0;
2012  user.rewardLockedUp = 0;
```

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/WhaleKiller.sol

Locations

```
2009  emit EmergencyWithdraw(msg.sender, _pid, user.amount);
2010  user.amount = 0;
2011  user.rewardDebt = 0;
2012  user.rewardLockedUp = 0;
2013  }
```

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/WhaleKiller.sol

Locations

```
2010  user.amount = 0;
2011  user.rewardDebt = 0;
2012  user.rewardLockedUp = 0;
2013  }
```

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/WhaleKiller.sol

Locations

```
1969 if (_amount > 0) {
1970 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
1971 if (address(pool.lpToken) == address(whale)) {
1972 uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
1973 _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/WhaleKiller.sol

Locations

```
1969 if (_amount > 0) {
1970 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
1971 if (address(pool.lpToken) == address(whale)) {
1972 uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
1973 _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/WhaleKiller.sol

Locations

```
1973 _amount = _amount.sub(transferTax);
1974 }
1975 if (pool.depositFeeBP > 0) {
1976 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1977 user.amount = user.amount.add(_amount).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/WhaleKiller.sol

Locations

```
1978 pool.lpToken.safeTransfer(feeAddress, depositFee);
1979 } else {
1980 user.amount = user.amount.add(_amount);
1981 }
1982 }
```

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/WhaleKiller.sol

Locations

```
1978 pool.lpToken.safeTransfer(feeAddress, depositFee);
1979 } else {
1980 user.amount = user.amount.add(_amount);
1981 }
1982 }
```

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/WhaleKiller.sol

Locations

```
1982 }
1983
1984 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
1985 emit Deposit(msg.sender, _pid, _amount);
1986 }
```

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/WhaleKiller.sol

Locations

```
1982 }
1983
1984 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
1985 emit Deposit(msg.sender, _pid, _amount);
1986 }
```

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/WhaleKiller.sol

Locations

```
1982 }
1983
1984 user.rewardDebt = user.amount.mul(pool.accWhalePerShare).div(1e12);
1985 emit Deposit(msg.sender, _pid, _amount);
1986 }
```

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/WhaleKiller.sol

Locations

```
1155 // By storing the original value once again, a refund is triggered (see
1156 // https://eips.ethereum.org/EIPS/eip-2200)
1157 _status = _NOT_ENTERED;
1158 }
1159 }
```

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/WhaleKiller.sol

Locations

```
1596 returns (uint256)
1597 {
1598 require(blockNumber < block.number, "WHALE::getPriorVotes: not yet determined");
1599
1600 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/WhaleKiller.sol

Locations

```
1669 internal
1670 {
1671 uint32 blockNumber = safe32(block.number, "WHALE::_writeCheckpoint: block number exceeds 32 bits");
1672
1673 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/WhaleKiller.sol

Locations

```
1835 massUpdatePools();
1836 }
1837 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1838 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1839 poolExistence[_lpToken] = true;
```

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/WhaleKiller.sol

Locations

```
1835 massUpdatePools();
1836 }
1837 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1838 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1839 poolExistence[_lpToken] = true;
```

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/WhaleKiller.sol

Locations

```
1887 uint256 accWhalePerShare = pool.accWhalePerShare;
1888 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1889 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1890 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1891 uint256 whaleReward = multiplier.mul(whalePerBlock).mul(pool.allocPoint).div(
```

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/WhaleKiller.sol

Locations

```
1888 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1889 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1890 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1891 uint256 whaleReward = multiplier.mul(whalePerBlock).mul(pool.allocPoint).div(
1892 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/WhaleKiller.sol

Locations

```
1911 function updatePool(uint256 _pid) public {
1912 PoolInfo storage pool = poolInfo[_pid];
1913 if (block.number <= pool.lastRewardBlock) {
1914 return;
1915 }
```

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/WhaleKiller.sol

Locations

```
1916 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1917 if (lpSupply == 0 || pool.allocPoint == 0) {
1918 pool.lastRewardBlock = block.number;
1919 return;
1920 }
```

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/WhaleKiller.sol

Locations

```
1919 return;
1920 }
1921 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1922 uint256 whaleReward =
1923 multiplier.mul(whalePerBlock).mul(pool.allocPoint).div(
```

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/WhaleKiller.sol

Locations

```
1929  whaleReward.mul(1e12).div(lpSupply)
1930  );
1931  pool.lastRewardBlock = block.number;
1932  }
```

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/WhaleKiller.sol

Locations

```
2021  uint256 totalRewards = pending.add(user.rewardLockedUp);
2022  uint256 lastBlockHarvest = startBlockHarvest.add(harvestTime);
2023  if (block.number >= startBlockHarvest && block.number <= lastBlockHarvest) {
2024  if (pending > 0 || user.rewardLockedUp > 0) {
2025  // reset lockup
```

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/WhaleKiller.sol

Locations

```
2021  uint256 totalRewards = pending.add(user.rewardLockedUp);
2022  uint256 lastBlockHarvest = startBlockHarvest.add(harvestTime);
2023  if (block.number >= startBlockHarvest && block.number <= lastBlockHarvest) {
2024  if (pending > 0 || user.rewardLockedUp > 0) {
2025  // reset lockup
```

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/WhaleKiller.sol

Locations

```
1914 return;
1915 }
1916 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1917 if (lpSupply == 0 || pool.allocPoint == 0) {
1918 pool.lastRewardBlock = block.number;
```

Source file

contracts/WhaleKiller.sol

Locations

```
1700 //
1701 // Have fun reading it. Hopefully it's bug-free. God bless.
1702 contract WhaleKiller is Ownable, ReentrancyGuard {
1703 using SafeMath for uint256;
1704 using SafeBEP20 for IBEP20;
1705
1706 // Info of each user.
1707 struct UserInfo {
1708 uint256 amount; // How many LP tokens the user has provided.
1709 uint256 rewardDebt; // Reward debt. See explanation below.
1710 uint256 rewardLockedUp; // Reward locked up.
1711 //
1712 // We do some fancy math here. Basically, any point in time, the amount of WHALEs
1713 // entitled to a user but is pending to be distributed is:
1714 //
1715 // pending reward = (user.amount * pool.accWhalePerShare) - user.rewardDebt
1716 //
1717 // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
1718 // 1. The pool's `accWhalePerShare` (and `lastRewardBlock`) gets updated.
1719 // 2. User receives the pending reward sent to his/her address.
1720 // 3. User's `amount` gets updated.
1721 // 4. User's `rewardDebt` gets updated.
1722 }
1723
1724 // Info of each pool.
1725 struct PoolInfo {
1726 IBEP20 lpToken; // Address of LP token contract.
1727 uint256 allocPoint; // How many allocation points assigned to this pool. WHALEs to distribute per block.
1728 uint256 lastRewardBlock; // Last block number that WHALEs distribution occurs.
1729 uint256 accWhalePerShare; // Accumulated WHALEs per share, times 1e12. See below.
1730 uint16 depositFeeBP; // Deposit fee in basis points
1731 }
1732
1733 // The WHALE Token!
1734 WhaleToken public whale;
1735 // Dev address.
1736 address public devAddr;
1737 // WHALE tokens created per block.
1738 uint256 public whalePerBlock;
1739 // Deposit Fee address
1740 address public feeAddress;
1741
1742 // Harvest time (how many block);
1743 uint256 public harvestTime;
1744 // Start Block Harvest
```

```
uint256 public startBlockHarvest;

// 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 WHALE mining starts.
uint256 public startBlock;
// Total locked up rewards
uint256 public totalLockedUpRewards;

// Referral Bonus in basis points. Initially set to 3%
uint256 public refBonusBP = 300;
// Max deposit fee capped to: 4%.
uint16 public constant MAXIMUM_DEPOSIT_FEE_BP = 400;
// Max referral commission rate: 10%.
uint16 public constant MAXIMUM_REFERRAL_BP = 1000;
// Referral Mapping
mapping(address => address) public referrers; // account_address -> referrer_address
mapping(address => uint256) public referredCount; // referrer_address -> num_of_referred
// Pool Exists Mapper
mapping(IBEP20 => bool) public poolExistence;
// Pool ID Tracker Mapper
mapping(IBEP20 => uint256) public poolIdForLpAddress;

// Initial emission rate: 1 WHALE per block.
uint256 public constant INITIAL_EMISSION_RATE = 10 ether;

// Initial harvest time: 1 day.
uint256 public constant INITIAL_HARVEST_TIME = 28800;

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 SetFeeAddress(address indexed user, address indexed _devAddress);
event SetDevAddress(address indexed user, address indexed _feeAddress);
event Referral(address indexed _referrer, address indexed _user);
event ReferralPaid(address indexed _user, address indexed _userTo, uint256 _reward);
event ReferralBonusBpChanged(uint256 _oldBp, uint256 _newBp);
event EmissionRateUpdated(address indexed caller, uint256 previousAmount, uint256 newAmount);
event UpdateHarvestTime(address indexed caller, uint256 _oldHarvestTime, uint256 _newHarvestTime);
event UpdateStartBlockHarvest(address indexed caller, uint256 _oldStartBlockHarvest, uint256 _newStartBlockHarvest);
event RewardLockedUp(address indexed user, uint256 indexed pid, uint256 amountLockedUp);

constructor(
WhaleToken _whale,
address _devAddr,
address _feeAddress,
uint256 _startBlock
) public {
whale = _whale;
devAddr = _devAddr;
feeAddress = _feeAddress;
whalePerBlock = INITIAL_EMISSION_RATE;
harvestTime = INITIAL_HARVEST_TIME;
startBlock = _startBlock;
startBlockHarvest = _startBlock + 144000;
```

```
}

// Get number of pools added.
function poolLength() external view returns (uint256) {
return poolInfo.length;
}

function getPoolIdForLpToken(IBEP20 _lpToken) external view returns (uint256) {
require(poolExistence[_lpToken] != false, "getPoolIdForLpToken: do not exist");
return poolIdForLpAddress[_lpToken];
}

// Modifier to check Duplicate pools
modifier nonDuplicated(IBEP20 _lpToken) {
require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");
_;
}

// 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 <= MAXIMUM_DEPOSIT_FEE_BP, "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,
accWhalePerShare: 0,
depositFeeBP: _depositFeeBP
})
);
poolIdForLpAddress[_lpToken] = poolInfo.length - 1;
}

// Update the given pool's WHALE 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 <= MAXIMUM_DEPOSIT_FEE_BP, "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
pure
returns (uint256)
{
return _to.sub(_from);
}

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

// 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];
if (block.number <= pool.lastRewardBlock) {
return;
}
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (lpSupply == 0 || pool.allocPoint == 0) {
pool.lastRewardBlock = block.number;
return;
}
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 whaleReward =
multiplier.mul(whalePerBlock).mul(pool.allocPoint).div(
totalAllocPoint
);
whale.mint(devAddr, whaleReward.div(10));
whale.mint(address(this), whaleReward);
pool.accWhalePerShare = pool.accWhalePerShare.add(
whaleReward.mul(1e12).div(lpSupply)
);
pool.lastRewardBlock = block.number;
}

```

```
// Deposit LP tokens to MasterChef for WHALE allocation.
function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
updatePool(_pid);
payOrLockupPendingWhale(_pid);
if (_amount > 0) {
pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
if (address(pool.lpToken) == address(whale)) {
uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
_amount = _amount.sub(transferTax);
}
if (pool.depositFeeBP > 0) {
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
user.amount = user.amount.add(_amount).sub(depositFee);
pool.lpToken.safeTransfer(feeAddress, depositFee);
} else {
user.amount = user.amount.add(_amount);
}
}

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

// Deposit LP tokens to MasterChef for WHALE allocation with referral.
function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
updatePool(_pid);
if (_amount > 0 && _referrer != address(0) && _referrer == address(_referrer) && _referrer != msg.sender) {
setReferral(msg.sender, _referrer);
}
payOrLockupPendingWhale(_pid);

if (_amount > 0) {
pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
if (address(pool.lpToken) == address(whale)) {
uint256 transferTax = _amount.mul(whale.transferTaxRate()).div(10000);
_amount = _amount.sub(transferTax);
}
if (pool.depositFeeBP > 0) {
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
user.amount = user.amount.add(_amount).sub(depositFee);
pool.lpToken.safeTransfer(feeAddress, depositFee);
} else {
user.amount = user.amount.add(_amount);
}
}

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

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

if (_amount > 0) {
```

```
user.amount = user.amount.sub(_amount);
pool.lpToken.safeTransfer(address(msg.sender), _amount);
}
user.rewardDebt = user.amount.mul(pool.accWhalePerShare).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];
pool.lpToken.safeTransfer(address(msg.sender), user.amount);
emit EmergencyWithdraw(msg.sender, _pid, user.amount);
user.amount = 0;
user.rewardDebt = 0;
user.rewardLockedUp = 0;
}

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

uint256 pending = user.amount.mul(pool.accWhalePerShare).div(1e12).sub(user.rewardDebt);
uint256 totalRewards = pending.add(user.rewardLockedUp);
uint256 lastBlockHarvest = startBlockHarvest.add(harvestTime);
if (block.number >= startBlockHarvest && block.number <= lastBlockHarvest) {
if (pending > 0 || user.rewardLockedUp > 0) {
// reset lockup
totalLockedUpRewards = totalLockedUpRewards.sub(user.rewardLockedUp);
user.rewardLockedUp = 0;

// send rewards
safeWhaleTransfer(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 whale transfer function, just in case if rounding error causes pool to not have enough WHALEs.
function safeWhaleTransfer(address _to, uint256 _amount) internal {
uint256 whaleBal = whale.balanceOf(address(this));
bool transferSuccess = false;
if (_amount > whaleBal) {
transferSuccess = whale.transfer(_to, whaleBal);
} else {
transferSuccess = whale.transfer(_to, _amount);
}
require(transferSuccess, "safeWhaleTransfer: transfer failed.");
}

// Update dev address by the previous dev.
function setDevAddress(address _devaddr) public {
require(_devaddr != address(0), "dev: invalid address");
require(msg.sender == devAddr, "dev: wut?");
devAddr = _devaddr;
emit SetDevAddress(msg.sender, _devaddr);
}

```

```
// Update fee address by the previous fee address.
function setFeeAddress(address _feeAddress) public {
require(_feeAddress != address(0), "setFeeAddress: invalid address");
require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
feeAddress = _feeAddress;
emit SetFeeAddress(msg.sender, _feeAddress);
}



// updateEmissionRate
function updateEmissionRate(uint256 _whalePerBlock) public onlyOwner {
massUpdatePools();
emit EmissionRateUpdated(msg.sender, whalePerBlock, _whalePerBlock);
whalePerBlock = _whalePerBlock;
}

// updateHarvestTime, how many blocks
function updateHarvestTime(uint256 _harvestTime) public onlyOwner {
harvestTime = _harvestTime;
emit UpdateHarvestTime(msg.sender, harvestTime, _harvestTime);
}

// updateStartBlockHarvest
function updateStartBlockHarvest(uint256 _startBlockHarvest) public onlyOwner {
startBlockHarvest = _startBlockHarvest;
emit UpdateStartBlockHarvest(msg.sender, startBlockHarvest, _startBlockHarvest);
}

// Set Referral Address for a user
function setReferral(address _user, address _referrer) internal {
if (_referrer == address(_referrer) && referrers[_user] == address(0) && _referrer != address(0) && _referrer != _user) {
referrers[_user] = _referrer;
referredCount[_referrer] += 1;
emit Referral(_user, _referrer);
}
}

// Get Referral Address for a Account
function getReferral(address _user) public view returns (address) {
return referrers[_user];
}

// Pay referral commission to the referrer who referred this user.
function payReferralCommission(address _user, uint256 _pending) internal {
address referrer = getReferral(_user);
if (referrer != address(0) && referrer != _user && refBonusBP > 0) {
uint256 refBonusEarned = _pending.mul(refBonusBP).div(10000);
whale.mint(referrer, refBonusEarned);
emit ReferralPaid(_user, referrer, refBonusEarned);
}
}

// Referral Bonus in basis points.
// Initially set to 3%, this gives the ability to increase or decrease the bonus referral percentage based on
// community voting and feedback.
function updateReferralBonusBp(uint256 _newRefBonusBp) public onlyOwner {
require(_newRefBonusBp <= MAXIMUM_REFERRAL_BP, "updateRefBonusPercent: invalid referral bonus basis points");
require(_newRefBonusBp != refBonusBP, "updateRefBonusPercent: same referral bonus set");
uint256 previousRefBonusBP = refBonusBP;
refBonusBP = _newRefBonusBp;
emit ReferralBonusBpChanged(previousRefBonusBP, _newRefBonusBp);
}
}
```