MythX

Analysis 1b888737-2ed9-4f84-9fbe-0cbb873c7121 MythX

Started	Wed Jun 02 2021 13:45:19 GMT+0000 (Coordinated Universal Time)
Finished	Wed Jun 02 2021 14:01:50 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts-01-06-21/Masterchef.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	37	32

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-01-06-21/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

SWC-000

Function could be marked as external.

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-01-06-21/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

SWC-000

Function could be marked as external.

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-01-06-21/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

SWC-000

Function could be marked as external.

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-01-06-21/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

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-01-06-21/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

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-01-06-21/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

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-01-06-21/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

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-01-06-21/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

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-01-06-21/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

Function could be marked as external.

SWC-000

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-01-06-21/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

Function could be marked as external.

SWC-000

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-01-06-21/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

Function could be marked as external.

SWC-000

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-01-06-21/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 OCTOPUS
```

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-01-06-21/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 octopusSwapRouter 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-01-06-21/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, "OCTOPUS::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-01-06-21/masterchef.sol

Locations

```
1538 * Can only be called by the current operator.
1539 */
1540 function updateBurnRate(uint16 _burnRate) public onlyOperator {
1541 require(_burnRate <= 100, "OCTOPUS::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-01-06-21/masterchef.sol

Locations

```
1548 * Can only be called by the current operator.
1549 */
1550 function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
1551 require(_maxTransferAmountRate <= 10000, "OCTOPUS::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-01-06-21/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-01-06-21/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-01-06-21/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 "updateOctopusSwapRouter" 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-01-06-21/masterchef.sol

Locations

```
1584 * Can only be called by the current operator.
1585 */
1586 function updateOctopusSwapRouter(address _router) public onlyOperator {
1587 octopusSwapRouter = IUniswapV2Router02(_router);
1588 octopusSwapPair = IUniswapV2Factory(octopusSwapRouter.factory()).getPair(address(this), octopusSwapRouter.WETH());
1589 require(octopusSwapPair != address(0), "OCTOPUS::updateOctopusSwapRouter: Invalid pair address.");
1590 emit OctopusSwapRouterUpdated(msg.sender, address(octopusSwapRouter), octopusSwapPair);
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-01-06-21/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), "OCTOPUS::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-01-06-21/masterchef.sol

Locations

```
1965 // Add a new lp to the pool. Can only be called by the owner.
1966 // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
1967 function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
1968 require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
1969 require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add: invalid harvest interval");
1970 if (_withUpdate) {
1971 massUpdatePools();
1972 }
1973 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1974 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1975 poolInfo.push(PoolInfo({
1976 lpToken: _lpToken,
1977 allocPoint: _allocPoint,
1978 lastRewardBlock: lastRewardBlock,
1979 accOctopusPerShare: 0,
1980 depositFeeBP: _depositFeeBP,
1981 harvestInterval: _harvestInterval
1982 }));
1983 }
1984
1985 // Add a new game to the pool. Can only be called by the owner.
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "addGame" 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-01-06-21/masterchef.sol

Locations

```
1985 // Add a new game to the pool. Can only be called by the owner.
1986 // XXX DO NOT add the same game address more than once and DO NOT add invalid address.
1987 function addGame(IBEP20 _game, uint256 _mintFeeBP) public onlyOwner {
1988 totalMintFeeForGame = totalMintFeeForGame.add(_mintFeeBP);
1989 require(totalMintFeeForGame <= 1000, "add: invalid mint fee basis points");
1990
1991 gameInfo.push(GameInfo({
1992 game: _game,
1993 mintFeeBP: _mintFeeBP
1994 }));
1995 }
1996
1997 /**
```

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-01-06-21/masterchef.sol

Locations

```
2002 * - `spender` cannot be the zero address.
2003 */
2004 function approve(address spender, uint256 amount) public onlyOwner returns (bool) {
2005 octopus.approve(spender, amount);
2006 return true;
2007 }
2008
2009 // Update the given pool's OCTOPUS 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-01-06-21/masterchef.sol

Locations

```
2008
2009 // Update the given pool's OCTOPUS allocation point and deposit fee. Can only be called by the owner.
2010 function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) public onlyOwner {
2011 require(_depositFeeBP <= 1000, "set: invalid deposit fee basis points");
2012 require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "set: invalid harvest interval");
2013 if (_withUpdate) {
2014 massUpdatePools();
2015 }
2016 totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
2017 poolInfo[_pid].allocPoint = _allocPoint;
2018 poolInfo[_pid].depositFeeBP = _depositFeeBP;
2019 poolInfo[_pid].harvestInterval = _harvestInterval;
2020 }
2021
2022 // Update the given gmaepool's minted fee. Can only be called by the owner.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setGame" 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-01-06-21/masterchef.sol

Locations

```
2021
2022  // Update the given gmaepool's minted fee. Can only be called by the owner.
2023  function setGame(uint256 _pid, uint16 _mintFeeBP) public onlyOwner {
2024  totalMintFeeForGame = totalMintFeeForGame.sub(gameInfo[_pid].mintFeeBP).add(_mintFeeBP);
2025  require(totalMintFeeForGame <= 10000, "set: invalid deposit fee basis points");
2026
2027  gameInfo[_pid].mintFeeBP = _mintFeeBP;
2028  }
2029
2030  // 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-01-06-21/masterchef.sol

Locations

```
2095
2096  // Deposit LP tokens to MasterChef for OCTOPUS allocation.
2097  function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
2098  PoolInfo storage pool = poolInfo[_pid];
2099  UserInfo storage user = userInfo[_pid][msg.sender];
2100  updatePool(_pid);
2101  if (_amount > 0 && address(octopusReferral) != address(0) && _referrer != address(0) && _referrer != msg.sender) {
2102  octopusReferral.recordReferral(msg.sender, _referrer);
2103  }
2104  payOrLockupPendingOctopus(_pid);
2105  if (_amount > 0) {
2106  pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2107  if (address(pool.lpToken) == address(octopus)) {
2108  uint256 transferTax = _amount.mul(octopus.transferTaxRate()).div(10000);
2109  _amount = _amount.sub(transferTax);
2110  }
2111  if (pool.depositFeeBP > 0) {
2112  uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2113  pool.lpToken.safeTransfer(feeAddBb, depositFee);
2114  user.amount = user.amount.add(_amount).sub(depositFee);
2115  pool.lpToken.safeTransfer(feeAddSt, depositFee);
2116  user.amount = user.amount.add(_amount).sub(depositFee);
2117  } else {
2118  user.amount = user.amount.add(_amount);
2119  }
2120  }
2121  user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2122  emit Deposit(msg.sender, _pid, _amount);
2123  }
2124
2125  // 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-01-06-21/masterchef.sol

Locations

```
2124
2125 // Withdraw LP tokens from MasterChef.
2126 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
2127 PoolInfo storage pool = poolInfo[_pid];
2128 UserInfo storage user = userInfo[_pid][msg.sender];
2129 require(user.amount >= _amount, "withdraw: not good");
2130 updatePool(_pid);
2131 payOrLockupPendingOctopus(_pid);
2132 if (_amount > 0) {
2133 user.amount = user.amount.sub(_amount);
2134 pool.lpToken.safeTransfer(address(msg.sender), _amount);
2135 }
2136 user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2137 emit Withdraw(msg.sender, _pid, _amount);
2138 }
2139
2140 // 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-01-06-21/masterchef.sol

Locations

```
2139
2140 // Withdraw without caring about rewards. EMERGENCY ONLY.
2141 function emergencyWithdraw(uint256 _pid) public nonReentrant {
2142 PoolInfo storage pool = poolInfo[_pid];
2143 UserInfo storage user = userInfo[_pid][msg.sender];
2144 uint256 amount = user.amount;
2145 user.amount = 0;
2146 user.rewardDebt = 0;
2147 user.rewardLockedUp = 0;
2148 user.nextHarvestUntil = 0;
2149 pool.lpToken.safeTransfer(address(msg.sender), amount);
2150 emit EmergencyWithdraw(msg.sender, _pid, amount);
2151 }
2152
2153 // Pay or lockup pending OCTOPUSs.
```

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-01-06-21/masterchef.sol

Locations

```
2192
2193  // Update dev address by the previous dev.
2194  function setDevAddress(address _devAddress) public {
2195  require(msg.sender == devAddress, "setDevAddress: FORBIDDEN");
2196  require(_devAddress != address(0), "setDevAddress: ZERO");
2197  devAddress = _devAddress;
2198  }
2199
2200  // 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-01-06-21/masterchef.sol

Locations

```
2199
2200  // Update marketing address by the previous marketing address.
2201  function setMarketingAddress(address _marketingAddress) public {
2202  require(msg.sender == marketingAddress, "setMarketingAddress: FORBIDDEN");
2203  require(_marketingAddress != address(0), "setMarketingAddress: ZERO");
2204  marketingAddress = _marketingAddress;
2205  }
2206
2207  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-01-06-21/masterchef.sol

Locations

```
2205  }
2206
2207  function setFeeAddrBaMa(address _feeAddBb) public {
2208  require(msg.sender == feeAddBb, "setFeeAddrBaMa: FORBIDDEN");
2209  require(_feeAddBb != address(0), "setFeeAddrBaMa: ZERO");
2210  feeAddBb = _feeAddBb;
2211  }
2212
2213  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-01-06-21/masterchef.sol

Locations

```
2211 }
2212
2213 function setFeeAddrStMa(address _feeAddSt) public {
2214 require(msg.sender == feeAddSt, "setFeeAddrStMa: FORBIDDEN");
2215 require(_feeAddSt != address(0), "setFeeAddrStMa: ZERO");
2216 feeAddSt = _feeAddSt;
2217 }
2218
2219 // 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-01-06-21/masterchef.sol

Locations

```
2218
2219 // Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
2220 function updateEmissionRate(uint256 _octopusPerBlock) public onlyOwner {
2221 emit EmissionRateUpdated(msg.sender, octopusPerBlock, _octopusPerBlock);
2222 octopusPerBlock = _octopusPerBlock;
2223 }
2224
2225 // Update the octopus referral contract address by the owner
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setOctopusReferral" 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-01-06-21/masterchef.sol

Locations

```
2224
2225 // Update the octopus referral contract address by the owner
2226 function setOctopusReferral(IOctopusReferral _octopusReferral) public onlyOwner {
2227 octopusReferral = _octopusReferral;
2228 }
2229
2230 // 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-01-06-21/masterchef.sol

Locations

```
2229
2230  // Update referral commission rate by the owner
2231  function setReferralCommissionRate(uint16 _referralCommissionRate) public onlyOwner {
2232  require(_referralCommissionRate <= MAXIMUM_REFERRAL_COMMISSION_RATE, "setReferralCommissionRate: invalid referral commission rate basis points");
2233  referralCommissionRate = _referralCommissionRate;
2234  }
2235
2236  // Pay referral commission to the referrer who referred this user.
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "updateMintBurnEnabled" 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-01-06-21/masterchef.sol

Locations

```
2252  * Can only be called by the owner.
2253  */
2254  function updateMintBurnEnabled(bool _enabled) public onlyOwner {
2255  emit MintBurnEnabledUpdated(msg.sender, _enabled);
2256  mintBurnEnabled = _enabled;
2257  }
2258
2259  }
```

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-01-06-21/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-01-06-21/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-01-06-21/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-01-06-21/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-01-06-21/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-01-06-21/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-01-06-21/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 ""^0.6.0"". 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-01-06-21/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 ""^0.6.0"". 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-01-06-21/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"". 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-01-06-21/masterchef.sol

Locations

```
1 // File: contracts/MasterChef.sol
2
3 pragma solidity ^0.6.0;
4
5 // MasterChef is the master of Cake. He can make Cake and he is a fair guy.
```

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-01-06-21/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-01-06-21/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-01-06-21/masterchef.sol

Locations

```
2105 if (_amount > 0) {
2106 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2107 if (address(pool.lpToken) == address(octopus)) {
2108 uint256 transferTax = _amount.mul(octopus.transferTaxRate()).div(10000);
2109 _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-01-06-21/masterchef.sol

Locations

```
2105 if (_amount > 0) {
2106 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2107 if (address(pool.lpToken) == address(octopus)) {
2108 uint256 transferTax = _amount.mul(octopus.transferTaxRate()).div(10000);
2109 _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-01-06-21/masterchef.sol

Locations

```
2109 _amount = _amount.sub(transferTax);
2110 }
2111 if (pool.depositFeeBP > 0) {
2112 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2113 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-01-06-21/masterchef.sol

Locations

```
2116 user.amount = user.amount.add(_amount).sub(depositFee);
2117 } else {
2118 user.amount = user.amount.add(_amount);
2119 }
2120 }
```

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-01-06-21/masterchef.sol

Locations

```
2116 | user.amount = user.amount.add(_amount).sub(depositFee);
2117 | } else {
2118 | user.amount = user.amount.add(_amount);
2119 | }
2120 | }
```

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-01-06-21/masterchef.sol

Locations

```
2119 | }
2120 | }
2121 | user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2122 | emit Deposit(msg.sender, _pid, _amount);
2123 | }
```

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-01-06-21/masterchef.sol

Locations

```
2119 | }
2120 | }
2121 | user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2122 | emit Deposit(msg.sender, _pid, _amount);
2123 | }
```

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-01-06-21/masterchef.sol

Locations

```
2119 }
2120 }
2121 user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2122 emit Deposit(msg.sender, _pid, _amount);
2123 }
```

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-01-06-21/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-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-01-06-21/masterchef.sol

Locations

```
1742 returns (uint256)
1743 {
1744 require(blockNumber < block.number, "OCTOPUS::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-01-06-21/masterchef.sol

Locations

```
1815 internal
1816 {
1817 uint32 blockNumber = safe32(block.number, "OCTOPUS::_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-01-06-21/masterchef.sol

Locations

```
1971 massUpdatePools();
1972 }
1973 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1974 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1975 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-01-06-21/masterchef.sol

Locations

```
1971 massUpdatePools();
1972 }
1973 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1974 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1975 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-01-06-21/masterchef.sol

Locations

```
2039 uint256 accOctopusPerShare = pool.accOctopusPerShare;
2040 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2041 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
2042 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
2043 uint256 octopusReward = multiplier.mul(octopusPerBlock).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-01-06-21/masterchef.sol

Locations

```
2040 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2041 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
2042 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
2043 uint256 octopusReward = multiplier.mul(octopusPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
2044 accOctopusPerShare = accOctopusPerShare.add(octopusReward.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-01-06-21/masterchef.sol

Locations

```
2065 function updatePool(uint256 _pid) public {
2066 PoolInfo storage pool = poolInfo[_pid];
2067 if (block.number <= pool.lastRewardBlock) {
2068 return;
2069 }
```

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-01-06-21/masterchef.sol

Locations

```
2070 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
2071 if (lpSupply == 0 || pool.allocPoint == 0) {
2072 pool.lastRewardBlock = block.number;
2073 return;
2074 }
```

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-01-06-21/masterchef.sol

Locations

```
2073 return;
2074 }
2075 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
2076 uint256 octopusReward = multiplier.mul(octopusPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
2077 octopus.mint(devAddress, octopusReward.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-01-06-21/masterchef.sol

Locations

```
2091 }
2092 pool.accOctopusPerShare = pool.accOctopusPerShare.add(octopusReward.mul(1e12).div(lpSupply));
2093 pool.lastRewardBlock = block.number;
2094 }
2095
```

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-01-06-21/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 }
```

Source file

/contracts-01-06-21/masterchef.sol

Locations

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

```
// Dev address.
address public devAddress;
// Marketing address.
address public marketingAddress;
// Deposit Fee address
address public feeAddBb;
address public feeAddSt;
// OCTOPUS tokens created per block.
uint256 public octopusPerBlock;
// Bonus muliplier for early octopus 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 OCTOPUS mining starts.
uint256 public startBlock;
// Total locked up rewards
uint256 public totalLockedUpRewards;

// Info of each pool.
GameInfo[] public gameInfo;
// Total allocation points. Must be the sum of all allocation points in all pools.
uint256 public totalMintFeeForGame = 0;

bool public mintBurnEnabled = true;

// Octopus referral contract address.
IOctopusReferral public octopusReferral;
// Referral commission rate in basis points.
uint16 public referralCommissionRate = 100;
// 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);
event MintBurnEnabledUpdated(address indexed operator, bool enabled);

constructor(
OctopusToken _octopus,
uint256 _startBlock,
uint256 _octopusPerBlock
) public {
octopus = _octopus;
startBlock = _startBlock;
octopusPerBlock = _octopusPerBlock;

devAddress = 0xd988BE748fbcEf8C04aA60646D94A31665fA6bc1;
marketingAddress = 0xAA5434d6f5ac5b78Ba7b3297a8fA46c46D573eC8;

feeAddBb = 0xCA59D5580B27666C00a96bfc3dC1C4CA6cC4f265;
feeAddSt = 0xb4308E3651155864Ce441647Fcafe233c6213B80;
```

```
}

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

function gameLength() external view returns (uint256) {
return gameInfo.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,
accOctopusPerShare: 0,
depositFeeBP: _depositFeeBP,
harvestInterval: _harvestInterval
}));
}

// Add a new game to the pool. Can only be called by the owner.
// XXX DO NOT add the same game address more than once and DO NOT add invalid address.
function addGame(IBEP20 _game, uint256 _mintFeeBP) public onlyOwner {
totalMintFeeForGame = totalMintFeeForGame.add(_mintFeeBP);
require(totalMintFeeForGame <= 1000, "add: invalid mint fee basis points");

gameInfo.push(GameInfo({
game: _game,
mintFeeBP: _mintFeeBP
}));
}

/**
* @dev See {BEP20-approve}.
*
* Requirements:
*
* - `spender` cannot be the zero address.
*/
function approve(address spender, uint256 amount) public onlyOwner returns (bool) {
octopus.approve(spender, amount);
return true;
}

// Update the given pool's OCTOPUS 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 <= 1000, "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;
}

// Update the given gmaepool's minted fee. Can only be called by the owner.
function setGame(uint256 _pid, uint16 _mintFeeBP) public onlyOwner {
totalMintFeeForGame = totalMintFeeForGame.sub(gameInfo[_pid].mintFeeBP).add(_mintFeeBP);
require(totalMintFeeForGame <= 10000, "set: invalid deposit fee basis points");

gameInfo[_pid].mintFeeBP = _mintFeeBP;
}

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

// View function to see if user can harvest OCTOPUSs.
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];
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 octopusReward = multiplier.mul(octopusPerBlock).mul(pool.allocPoint).div(totalAllocPoint);

```

```
2080 octopus.mint(devAddress, octopusReward.div(30));
2081 octopus.mint(marketingAddress, octopusReward.mul(2).div(30));
2082 if (mintBurnEnabled) {
2083 octopus.mint(address(this), octopusReward.mul(90).div(100));
2084
2085 uint256 length = gameInfo.length;
2086 for (uint256 pid = 0; pid < length; ++pid) {
2087 octopus.mint(address(gameInfo[pid].game), octopusReward.mul(gameInfo[pid].mintFeeBP).div(10000));
2088 }
2089
2090 uint256 mintBurnFee = 1000 - totalMintFeeForGame;
2091 octopus.mint(address(0), octopusReward.mul(mintBurnFee).div(10000));
2092 } else {
2093 octopus.mint(address(this), octopusReward);
2094 }
2095 pool.accOctopusPerShare = pool.accOctopusPerShare.add(octopusReward.mul(1e12).div(lpSupply));
2096 pool.lastRewardBlock = block.number;
2097 }
2098
2099 // Deposit LP tokens to MasterChef for OCTOPUS allocation.
2100 function deposit(uint256 _pid, uint256 _amount, address _referrer) public nonReentrant {
2101 PoolInfo storage pool = poolInfo[_pid];
2102 UserInfo storage user = userInfo[_pid][msg.sender];
2103 updatePool(_pid);
2104 if (_amount > 0 && address(octopusReferral) != address(0) && _referrer != address(0) && _referrer != msg.sender) {
2105 octopusReferral.recordReferral(msg.sender, _referrer);
2106 }
2107 payOrLockupPendingOctopus(_pid);
2108 if (_amount > 0) {
2109 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
2110 if (address(pool.lpToken) == address(octopus)) {
2111 uint256 transferTax = _amount.mul(octopus.transferTaxRate()).div(10000);
2112 _amount = _amount.sub(transferTax);
2113 }
2114 if (pool.depositFeeBP > 0) {
2115 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(2).div(10000);
2116 pool.lpToken.safeTransfer(feeAddBb, depositFee);
2117 user.amount = user.amount.add(_amount).sub(depositFee);
2118 pool.lpToken.safeTransfer(feeAddSt, depositFee);
2119 user.amount = user.amount.add(_amount).sub(depositFee);
2120 } else {
2121 user.amount = user.amount.add(_amount);
2122 }
2123 }
2124 user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2125 emit Deposit(msg.sender, _pid, _amount);
2126 }
2127
2128 // Withdraw LP tokens from MasterChef.
2129 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
2130 PoolInfo storage pool = poolInfo[_pid];
2131 UserInfo storage user = userInfo[_pid][msg.sender];
2132 require(user.amount >= _amount, "withdraw: not good");
2133 updatePool(_pid);
2134 payOrLockupPendingOctopus(_pid);
2135 if (_amount > 0) {
2136 user.amount = user.amount.sub(_amount);
2137 pool.lpToken.safeTransfer(address(msg.sender), _amount);
2138 }
2139 user.rewardDebt = user.amount.mul(pool.accOctopusPerShare).div(1e12);
2140 emit Withdraw(msg.sender, _pid, _amount);
```

```
}

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
uint256 amount = user.amount;
user.amount = 0;
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 OCTOPUSs.
function payOrLockupPendingOctopus(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.accOctopusPerShare).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
safeOctopusTransfer(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 octopus transfer function, just in case if rounding error causes pool to not have enough OCTOPUSs.
function safeOctopusTransfer(address _to, uint256 _amount) internal {
uint256 octopusBal = octopus.balanceOf(address(this));
if (_amount > octopusBal) {
octopus.transfer(_to, octopusBal);
} else {
octopus.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;
}
```

```
2203
2204 // Update marketing address by the previous marketing address.
2205 function setMarketingAddress(address _marketingAddress) public {
2206 require(msg.sender == marketingAddress, "setMarketingAddress: FORBIDDEN");
2207 require(_marketingAddress != address(0), "setMarketingAddress: ZERO");
2208 marketingAddress = _marketingAddress;
2209 }
2210
2211 function setFeeAddrBaMa(address _feeAddBb) public {
2212 require(msg.sender == feeAddBb, "setFeeAddrBaMa: FORBIDDEN");
2213 require(_feeAddBb != address(0), "setFeeAddrBaMa: ZERO");
2214 feeAddBb = _feeAddBb;
2215 }
2216
2217 function setFeeAddrStMa(address _feeAddSt) public {
2218 require(msg.sender == feeAddSt, "setFeeAddrStMa: FORBIDDEN");
2219 require(_feeAddSt != address(0), "setFeeAddrStMa: ZERO");
2220 feeAddSt = _feeAddSt;
2221 }
2222
2223 // Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
2224 function updateEmissionRate(uint256 _octopusPerBlock) public onlyOwner {
2225 emit EmissionRateUpdated(msg.sender, octopusPerBlock, _octopusPerBlock);
2226 octopusPerBlock = _octopusPerBlock;
2227 }
2228
2229 // Update the octopus referral contract address by the owner
2230 function setOctopusReferral(IOctopusReferral _octopusReferral) public onlyOwner {
2231 octopusReferral = _octopusReferral;
2232 }
2233
2234 // Update referral commission rate by the owner
2235 function setReferralCommissionRate(uint16 _referralCommissionRate) public onlyOwner {
2236 require(_referralCommissionRate <= MAXIMUM_REFERRAL_COMMISSION_RATE, "setReferralCommissionRate: invalid referral commission rate basis points");
2237 referralCommissionRate = _referralCommissionRate;
2238 }
2239
2240 // Pay referral commission to the referrer who referred this user.
2241 function payReferralCommission(address _user, uint256 _pending) internal {
2242 if (address(octopusReferral) != address(0) && referralCommissionRate > 0) {
2243 address referrer = octopusReferral.getReferrer(_user);
2244 uint256 commissionAmount = _pending.mul(referralCommissionRate).div(10000);
2245
2246 if (referrer != address(0) && commissionAmount > 0) {
2247 octopus.mint(referrer, commissionAmount);
2248 octopusReferral.recordReferralCommission(referrer, commissionAmount);
2249 emit ReferralCommissionPaid(_user, referrer, commissionAmount);
2250 }
2251 }
2252 }
2253
2254 /**
2255 * @dev Update the MintBurnEnabled.
2256 * Can only be called by the owner.
2257 */
2258 function updateMintBurnEnabled(bool _enabled) public onlyOwner {
2259 emit MintBurnEnabledUpdated(msg.sender, _enabled);
    mintBurnEnabled = _enabled;
    }


    }
```