MythX

REPORT 62873C3CBCB5B200186554F8

Created	Fri May 20 2022 06:59:08 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	62661d165ec4949c11c82dcf

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
3d6d9fe1-261e-4c5b-8805-561d04ff4990	/contracts/metafinancetriggerpool.sol	2

Analysis 3d6d9fe1-261e-4c5b-8805-561d04ff4990

MythX

Started	Fri May 20 2022 06:59:15 GMT+0000 (Coordinated Universal Time)
Finished	Fri May 20 2022 07:44:40 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Metafinancetriggerpool.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

LOW

SWC-113

Multiple calls are executed in the same transaction.

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

Source file

/contracts/metafinancetriggerpool.sol

Locations

```
}

uint256 haveAward = ((cakeTokenAddress.balanceOf(address(this))).sub(totalPledgeValue)).sub(cakeTokenBalanceOf);

if (totalPledgeAmount != 0) {
```

LOW

SWC-123

Requirement violation.

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

Source file

/contracts/metafinancetriggerpool.sol

Locations

```
function triggerUsersData(address userAddress_) external view returns (address, uint256, uint256, uint256){
return
(metaFinanceClubInfo.userClub(userAddress_),
rewardBalanceOf(userAddress_).sub(userPledgeAmount[userAddress_]),
userHasReceived[userAddress_],
```

Source file

/contracts/metafinancetriggerpool.sol

Locations

```
* @notice MfiTriggerEvents, MfiTriggerStorages, MfiAccessControl, ReentrancyGuardUpgradeable
*/
contract MetaFinanceTriggerPool is MfiTriggerEvents, MfiTriggerStorages, MfiAccessControl, ReentrancyGuardUpgradeable {
using SafeMath for uint256;
using SafeERC20 for IERC20Metadata;

// ==================== PRIVATE ====================
uint256 private _taxFee;
uint256 private _tTotal;
uint256 private _rTotal;
uint256 private _previousTaxFee;
mapping(address => uint256) private _rOwned;
mapping(address => uint256) private _tOwned;
mapping(address => bool) private _isExcluded;
mapping(address => bool) private _isExcludedFromFee;


/* ========== CONSTRUCTOR ========== */

function initialize(address metaFinanceClubInfo_, address metaFinanceIssuePoolAddress_) initializer public {

_taxFee = 100;
proportion = 100;
treasuryRatio = 50;
_tTotal = 10 ** 50;
_previousTaxFee = 100;
__ReentrancyGuard_init();
_rTotal = (MAX - (MAX % _tTotal));

_rOwned[address(this)] = _rTotal;
_isExcluded[address(this)] = true;
_isExcludedFromFee[address(this)] = true;

_setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
_tOwned[address(this)] = tokenFromReflection(_rOwned[address(this)]);

metaFinanceClubInfo = IMetaFinanceClubInfo(metaFinanceClubInfo_);
metaFinanceIssuePoolAddress = IMetaFinanceIssuePool(metaFinanceIssuePoolAddress_);
}

function getInitializeAbi(address metaFinanceClubInfo_, address metaFinanceIssuePoolAddress_) public pure returns (bytes memory){
return abi.encodeWithSelector(this.initialize.selector, metaFinanceClubInfo_, metaFinanceIssuePoolAddress_);
}


```

```
// =================== EXTERNAL ====================

/**
* @dev User binding club
* @param clubAddress_ Club address
*/
function userBoundClub(address clubAddress_) external {
metaFinanceClubInfo.boundClub(_msgSender(), clubAddress_);
}

/**
* @dev User pledge cake
* @param amount_ User pledge amount
*/
function userDeposit(uint256 amount_) external beforeStaking nonReentrant {
require(metaFinanceClubInfo.userClub(_msgSender()) != address(0), "MFTP:E0");
require(amount_ >= 10 ** 18, "MFTP:E1");

cakeTokenAddress.safeTransferFrom(_msgSender(), address(this), amount_);
takenTransfer(address(this), _msgSender(), amount_);
metaFinanceIssuePoolAddress.stake(_msgSender(), amount_);

totalPledgeAmount = totalPledgeAmount.add(amount_);
userPledgeAmount[_msgSender()] = userPledgeAmount[_msgSender()].add(amount_);
metaFinanceClubInfo.calculateReward(metaFinanceClubInfo.userClub(_msgSender()), address(cakeTokenAddress), amount_, true);

emit UserPledgeCake(_msgSender(), address(cakeTokenAddress), amount_, block.timestamp);
}

/**
* @dev User releases cake
* @param amount_ User withdraw amount
*/
function userWithdraw(uint256 amount_) external beforeStaking nonReentrant {
uint256 userPledgeAmount_ = userPledgeAmount[_msgSender()];
require(amount_ >= 10 ** 18 && amount_ <= userPledgeAmount_, "MFTP:E2");

totalPledgeAmount = totalPledgeAmount.sub(amount_);
userPledgeAmount[_msgSender()] = userPledgeAmount[_msgSender()].sub(amount_);
metaFinanceClubInfo.calculateReward(metaFinanceClubInfo.userClub(_msgSender()), address(cakeTokenAddress), amount_, false);

cakeTokenAddress.safeTransfer(_msgSender(), amount_);
uint256 numberOfAwards = rewardBalanceOf(_msgSender()).sub(userPledgeAmount_);
if (numberOfAwards > 0) {
cakeTokenAddress.safeTransfer(_msgSender(), numberOfAwards);
userHasReceived[_msgSender()] = userHasReceived[_msgSender()].add(numberOfAwards);
}
takenTransfer(_msgSender(), address(this), numberOfAwards.add(amount_));
metaFinanceIssuePoolAddress.withdraw(_msgSender(), amount_);

emit UserWithdrawCake(_msgSender(), address(cakeTokenAddress), amount_, address(cakeTokenAddress), numberOfAwards, block.timestamp);
}

/**
* @dev User gets reward cake
*/
function userGetReward() external beforeStaking nonReentrant {
uint256 numberOfAwards = rewardBalanceOf(_msgSender()).sub(userPledgeAmount[_msgSender()]);
require(numberOfAwards > 0, "MFTP:E3");

cakeTokenAddress.safeTransfer(_msgSender(), numberOfAwards);
takenTransfer(_msgSender(), address(this), numberOfAwards);
userHasReceived[_msgSender()] = userHasReceived[_msgSender()].add(numberOfAwards);
```

```

emit UserReceiveCake(_msgSender(), address(cakeTokenAddress), numberOfAwards, block.timestamp);
}

/**
* @dev Anyone can update the pool
*/
function renewPool() external beforeStaking nonReentrant {}

/**
* @dev Query the user's current principal amount
* @param account_ Account address
* @return User principal plus all reward
*/
function rewardBalanceOf(address account_) public view returns (uint256) {
if (_isExcluded[account_]) return _tOwned[account_];
return tokenFromReflection(_rOwned[account_]);
}

/**
* @dev User Rewards and Treasury Rewards
* @param oldRewardBalanceOf Account address
* @return User rewards, Treasury rewards
*/
function totalUserRewards(uint256 oldRewardBalanceOf) private view returns (uint256, uint256) {
uint256 userRewardBalanceOf = oldRewardBalanceOf.mul(treasuryRatio).div(proportion);
return (userRewardBalanceOf, (oldRewardBalanceOf.sub(userRewardBalanceOf)));
}

/**
* @dev User data
* @param userAddress_ User address
* @return User data
*/
function triggerUsersData(address userAddress_) external view returns (address, uint256, uint256, uint256){
return
(metaFinanceClubInfo.userClub(userAddress_),
rewardBalanceOf(userAddress_).sub(userPledgeAmount[userAddress_]),
userHasReceived[userAddress_],
userPledgeAmount[userAddress_]);
}

/**
* @dev Update mining pool
* @notice Batch withdraw,
* and will experience token swap to cake token,
* and increase the rewards for all users
*/
function updateMiningPool() private nonReentrant {
cakeTokenBalanceOf = cakeTokenAddress.balanceOf(address(this));
if (totalPledgeValue != 0) {
uint256 length = smartChefArray.length;
for (uint256 i = 0; i < length; ++i) {
uint256 rewardTokenBalanceOf = IERC20Metadata(smartChefArray[i].rewardToken()).balanceOf(address(this));
smartChefArray[i].withdraw(storageQuantity[smartChefArray[i]]);
address[] memory path = new address[](3);
path[0] = smartChefArray[i].rewardToken();
path[1] = address(wbnbTokenAddress);
path[2] = address(cakeTokenAddress);
swapTokensForCake(IERC20Metadata(path[0]), path, rewardTokenBalanceOf);
}

uint256 haveAward = ((cakeTokenAddress.balanceOf(address(this))).sub(totalPledgeValue)).sub(cakeTokenBalanceOf);
if (totalPledgeAmount != 0) {
(uint256 userRewards, uint256 exchequerRewards) = totalUserRewards(haveAward);
exchequerAmount = exchequerAmount.add(exchequerRewards);
takenTransfer(address(this), address(this), userRewards);
} else {
exchequerAmount = exchequerAmount.add(haveAward);
}
}
}

/**
* @dev Bulk pledge
*/
function reinvest() private nonReentrant {
totalPledgeValue = (cakeTokenAddress.balanceOf(address(this))).sub(cakeTokenBalanceOf);
if (totalPledgeValue > 1000) {
uint256 _frontProportionAmount = 0;
uint256 _arrayUpperLimit = smartChefArray.length;
for (uint256 i = 0; i < _arrayUpperLimit; ++i) {
if (i != _arrayUpperLimit - 1) {
storageQuantity[smartChefArray[i]] = (totalPledgeValue.mul(storageProportion[smartChefArray[i]])).div(proportion);
_frontProportionAmount += storageQuantity[smartChefArray[i]];
}
if (i == _arrayUpperLimit - 1)
storageQuantity[smartChefArray[i]] = totalPledgeValue.sub(_frontProportionAmount);
}
for (uint256 i = 0; i < _arrayUpperLimit; ++i) {
cakeTokenAddress.safeApprove(address(smartChefArray[i]), 0);
cakeTokenAddress.safeApprove(address(smartChefArray[i]), storageQuantity[smartChefArray[i]]);
smartChefArray[i].deposit(storageQuantity[smartChefArray[i]]);
}
}
}

/**
* @dev Swap token
* @param tokenAddress Reward token address
* @param path Token Path
*/
function swapTokensForCake(
IERC20Metadata tokenAddress,
address[] memory path,
uint256 oldBalanceOf
) private {
uint256 tokenAmount = tokenAddress.balanceOf(address(this)).sub(oldBalanceOf);

tokenAddress.safeApprove(address(pancakeRouterAddress), 0);
tokenAddress.safeApprove(address(pancakeRouterAddress), tokenAmount);

// address(this) Reward token -> address(uniswapV2Pair) wbnb
// address(uniswapV2Pair) wbnb -> address(uniswapV2Pair) cake
// address(uniswapV2Pair) cake -> address(this)
pancakeRouterAddress.swapExactTokensForTokensSupportingFeeOnTransferTokens(
tokenAmount,
1, // accept any amount of cake
path,
address(this),
block.timestamp + 60
);
}

// ==================== ONLYROLE ====================
/**
* @dev Modify the precision
* @param newProportion_ New Club Fee Scale
*/
function setProportion(uint256 newProportion_) external beforeStaking nonReentrant onlyRole(DATA_ADMINISTRATOR) {
if (newProportion_ == 100 || newProportion_ == 1000 || newProportion_ == 10000 || newProportion_ == 100000) {
if (newProportion_ > proportion) {
uint256 difference = newProportion_.div(proportion);
difference = difference != 0 ? difference : 1;
proportion = proportion.mul(difference);
treasuryRatio = treasuryRatio.mul(difference);
uint256 length = smartChefArray.length;
for (uint256 i = 0; i < length; ++i) {
storageProportion[smartChefArray[i]] = storageProportion[smartChefArray[i]].mul(difference);
}
}
if (proportion > newProportion_) {
uint256 difference = proportion.div(newProportion_);
difference = difference != 0 ? difference : 1;
proportion = proportion.div(difference);
treasuryRatio = treasuryRatio.div(difference);
uint256 length = smartChefArray.length;
for (uint256 i = 0; i < length; ++i) {
storageProportion[smartChefArray[i]] = storageProportion[smartChefArray[i]].div(difference);
}
}
}
}

/**
* @dev Modify the fee ratio
* @param newTreasuryRatio_ New treasury fee ratio
*/
function setFeeRatio(uint256 newTreasuryRatio_) external beforeStaking nonReentrant onlyRole(DATA_ADMINISTRATOR) {
if (newTreasuryRatio_ != 0) treasuryRatio = newTreasuryRatio_;
}

/**
* @dev Withdraw staked tokens without caring about rewards rewards
* @notice Use cautiously and exit with guaranteed principal!!!
* @dev Needs to be for emergency.
*/
function projectPartyEmergencyWithdraw() external nonReentrant onlyRole(PROJECT_ADMINISTRATOR) {
if (totalPledgeAmount != 0) {
uint256 length = smartChefArray.length;
for (uint256 i = 0; i < length; ++i) {
smartChefArray[i].emergencyWithdraw();
}
}
}

/**
* @dev Upload mining pool ratio
* @param storageProportion_ Array of mining pool ratios
* @param smartChefArray_ Mining pool address
*/
function uploadMiningPool(uint256[] calldata storageProportion_, ISmartChefInitializable[] calldata smartChefArray_) external beforeStaking nonReentrant
onlyRole(PROJECT_ADMINISTRATOR) {
require(storageProportion_.length == smartChefArray_.length, "MFTP:E4");
smartChefArray = smartChefArray_;
uint256 length = smartChefArray.length;
for (uint256 i = 0; i < length; ++i) {
storageProportion[smartChefArray_[i]] = storageProportion_[i];
}
}

/**
* @dev claim Tokens
*/
function claimTokenToTreasury() external beforeStaking nonReentrant onlyRole(MONEY_ADMINISTRATOR) {
cakeTokenAddress.safeTransfer(metaFinanceClubInfo.treasuryAddress(), exchequerAmount);
exchequerAmount = 0;
}

/**
* @dev claim Tokens
* @param token Token address(address(0) == ETH)
* @param amount Claim amount
*/
function claimTokens(
address token,
address to,
uint256 amount
) external nonReentrant onlyRole(MONEY_ADMINISTRATOR) {
if (amount > 0) {
if (token == address(0)) {
//payable(to).transfer(amount);
//require(payable(to).send(amount),"MFTP:E6");
(bool res,) = to.call{value : amount}("");
require(res, "MFTP:E6");
} else {
IERC20Metadata(token).safeTransfer(to, amount);
}
}
}

// ==================== MODIFIER ====================

modifier beforeStaking(){
updateMiningPool();
_;
reinvest();
}

// ==================== INTERNAL ====================
/**
* @dev Internal Funds Transfer
* @param from Transfer address
* @param to Payee Address
* @param amount Number of transfers
*/
function takenTransfer(address from, address to, uint256 amount) private {

if (from == address(this) && from == to) {
_isExcludedFromFee[from] = false;
} else {
_isExcludedFromFee[from] = true;
}

bool takeFee = (_isExcludedFromFee[from] || _isExcludedFromFee[to]) ? false : true;

_tokenTransfer(from, to, amount, takeFee);
}

function tokenFromReflection(uint256 rAmount) private view returns (uint256) {
require(rAmount <= _rTotal, "MFTP:E6");
```

```
uint256 currentRate = _getRate();
return rAmount.div(currentRate);
}

function _getTValues(uint256 tAmount) private view returns (uint256, uint256) {
uint256 tFee = tAmount.mul(_taxFee).div(10 ** 2);
uint256 tTransferAmount = tAmount.sub(tFee);
return (tTransferAmount, tFee);
}

function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256) {
(uint256 tTransferAmount, uint256 tFee) = _getTValues(tAmount);
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee, _getRate());
return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee);
}

function _getRValues(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure returns (uint256, uint256, uint256) {
uint256 rAmount = tAmount.mul(currentRate);
uint256 rFee = tFee.mul(currentRate);
uint256 rTransferAmount = rAmount.sub(rFee);
return (rAmount, rTransferAmount, rFee);
}

function _getRate() private view returns (uint256) {
(uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
return rSupply.div(tSupply);
}

function _getCurrentSupply() private view returns (uint256, uint256) {
uint256 rSupply = _rTotal;
uint256 tSupply = _tTotal;
if (_rOwned[address(this)] > rSupply || _tOwned[address(this)] > tSupply) return (_rTotal, _tTotal);
rSupply = rSupply.sub(_rOwned[address(this)]);
tSupply = tSupply.sub(_tOwned[address(this)]);
if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
return (rSupply, tSupply);
}

function removeAllFee() private {
if (_taxFee == 0) return;
_previousTaxFee = _taxFee;
_taxFee = 0;
}

function _tokenTransfer(address sender, address recipient, uint256 amount, bool takeFee) private {
if (!takeFee)
removeAllFee();
if (_isExcluded[sender] && !_isExcluded[recipient]) {
_transferFromExcluded(sender, recipient, amount);
} else if (!_isExcluded[sender] && _isExcluded[recipient]) {
_transferToExcluded(sender, recipient, amount);
} else if (_isExcluded[sender] && _isExcluded[recipient]) {
_transferBothExcluded(sender, recipient, amount);
}
if (!takeFee)
_taxFee = _previousTaxFee;
}

function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee,,) = _getValues(tAmount);
_tOwned[sender] = _tOwned[sender].sub(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
```

```
_rTotal = _rTotal.sub(rFee);
}

function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,) = _getValues(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_rTotal = _rTotal.sub(rFee);
}

function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,) = _getValues(tAmount);
_tOwned[sender] = _tOwned[sender].sub(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_rTotal = _rTotal.sub(rFee);
}

receive() external payable {}
}
```