Competitive Security Assessment

Magpie_ArbStreaming

Jan 5th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

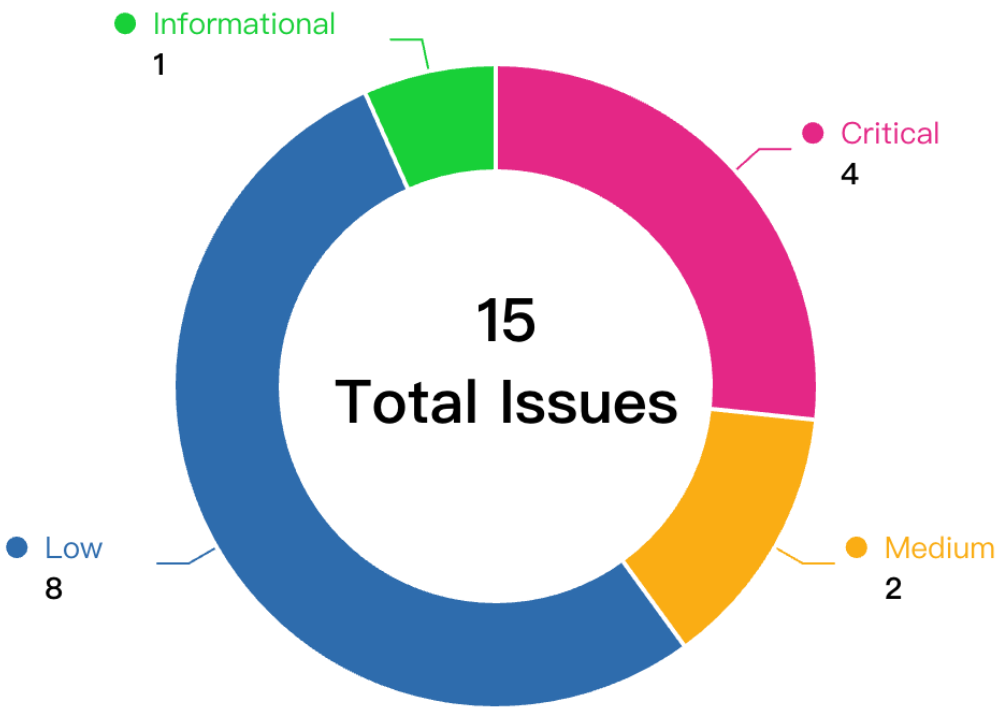
Project Detail

Project Name	Magpie_ArbStreaming
Platform & Language	Solidity
Codebase	• https://github.com/magpiexyz/magpie_contracts/pull/158 • audit commit - 0411a32dc88df0ce75727ba7afd5f67a6ca9d826 • final commit - 5f9f269e2169b5cbc1b7cd1a87f28a0ca344c0e3 • https://github.com/magpiexyz/penpie-contracts/pull/119 • audit commit - 68268b275419b96ed3826a19f3dfc00f4f9651f9 • final commit - 58aad544333ae6fcc2448b4eb29e6b72cec6626a • https://github.com/magpiexyz/radpie_contracts/pull/75 • audit commit - fee9cd3f3482ad3e3761cf8781b1bb63e3acf652 • final commit - 66d5fa467a5c59c52a7af65229360276b4b5dcd5
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
magpie_contracts/contracts/rewards/MasterMagpie.sol	**555ba4902fa086d914b613e1a264dcdae8a7b4d8d93618446921ab582f3d0a98**
magpie_contracts/contracts/MagpieReader.sol	**d19a7f1721c789b83859c1ce30c89cfe57264f490157247fdc86221e8b023e9b**
magpie_contracts/contracts/MagpieReaderEth.sol	**00a25df7cf51c43eb5f505c6ebce61fb1e525d6ac05993aa904558dd8349f816**
magpie_contracts/contracts/MagpieReaderArb.sol	**4fae5f395d8da563f588c5b1eae9ca25c093dbd94c987db5e83eb6adcd1f5d77**
magpie_contracts/contracts/interfaces/IBaseRewardPool.sol	**54bd4f4b7f64f67828196ac0680fc372825db7b944bc9a72c49618d9ccd7dafd**
magpie_contracts/contracts/interfaces/IARBRewarder.sol	**9d13bcdf30943782a9b0a34d35e0a52c36dca2895264c64536f00dbe71d62adb**
penpie-contracts/contracts/rewards/MasterPenpie.sol	**8d23d2a0d1af18e4dd53145fff7870ba84449de2086bd1c6f9d3c50781c5e6f6**
penpie-contracts/contracts/rewards/ARBRewarder.sol	**e78c2302af72f97cd0fe03848a138113c3b0557afe97be4ca2a4abd1a2f8216d**
penpie-contracts/contracts/interfaces/IMasterPenpie.sol	**c20592a1c1d53faf91ac6798ea190ab3577b1fe87b078e129d5f9d6186787745**
penpie-contracts/contracts/interfaces/IBaseRewardPool.sol	**b335ca8f69d6c6f183903eb23c6196073c1ff89790030b5a11c9bf784f738a03**
penpie-contracts/contracts/interfaces/IARBRewarder.sol	**8e184973e73574903b888b7618a8f3d35420b6ca321e0f9d98c3907c998be07e**
radpie_contracts/contracts/rewards/MasterRadpie.sol	**6c76910b29bc0a230574d1d8f03121df7aa83a91d19a761bf1a2c7bc50ba6fdf**
radpie_contracts/contracts/interfaces/IBaseRewardPool.sol	**9d425a6ae50b0d6a34366cc17b9fc96c2580be84062925818ef03593725ea26f**
radpie_contracts/contracts/interfaces/IARBRewarder.sol	**9d13bcdf30943782a9b0a34d35e0a52c36dca2895264c64536f00dbe71d62adb**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MPA-1	ARBReward Precision Loss	Logical	Critical	Fixed	biakia, rajatbeladiya
MPA-2	Unclaimed Arb rewards will not be sent to the user	Logical	Critical	Fixed	biakia

MPA-3	The `depositMPendleSVFor` and `depositVlPenpieFor` functions did not update ARBRewarder's debt, causing the user to get more arb rewards than expected	Logical	Critical	Fixed	biakia
MPA-4	The utilization of the function `_getUserStaked()` in the contract is vulnerable to sandwich attack.	Logical	Critical	Acknowledged	n16h7m4r3
MPA-5	`getTokenPrice` will return wrong price when chainlink usd price feed's `decimal != 8`	Logical	Medium	Acknowledged	biakia
MPA-6	Chainlink's latestRoundData might return stale results	Oracle Manipulation	Medium	Acknowledged	biakia, rajatbeladiya
MPA-7	potential DOS attack in `MasterMagpie::_sendARBRewards` function	DOS	Low	Acknowledged	ginlee
MPA-8	Missing check Arbitrum sequencer status when fetching prices from Chainlink feed	Oracle Manipulation	Low	Acknowledged	biakia
MPA-9	Not check whether the pool exists when updating `allocPoint`	Logical	Low	Fixed	biakia
MPA-10	There is no function to initialize `MagpieReaderArb`	Logical	Low	Acknowledged	biakia
MPA-11	Use `disableInitializers` to prevent any future reinitialization	Code Style	Low	Acknowledged	biakia
MPA-12	The function `pendingARB` maybe use an incorrect `_masterChef` address	Logical	Low	Fixed	biakia
MPA-13	`massUpdatePools()` is susceptible to DoS with block gas limit	Logical	Low	Mitigated	thereksfour, rajatbeladiya
MPA-14	Make `AllocationManagers` in MasterMagpie.sol as mapping instead of array as it may cause DOS by unbounded loop	DOS	Low	Acknowledged	grep-er

MPA-15	**No need to use SafeMath in solidity version 0.8+**	**Language Specific**	**Informational**	**Fixed**	**biakia**

MPA-1:ARBReward Precision Loss

Category	Severity	Client Response	Contributor
Logical	**Critical**	**Fixed**	**biakia, rajatbeladiya**

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L295-L317
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L306-L308

```
function _updatePool(address _stakingToken, uint256 _totalStaked) internal {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
        if (block.timestamp <= pool.lastRewardTimestamp || totalAllocPoint == 0)
            return;

        if (_totalStaked == 0) {
            pool.lastRewardTimestamp = block.timestamp;
            return;
        }

        uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
        uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;

        pool.accARBPerShare = pool.accARBPerShare + ((ARBReward * 1e12) / _totalStaked);
        pool.lastRewardTimestamp = block.timestamp;

        emit UpdatePool(
            _stakingToken,
            pool.lastRewardTimestamp,
            _totalStaked,
            pool.accARBPerShare
        );
    }

uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;

        pool.accARBPerShare = pool.accARBPerShare + ((ARBReward * 1e12) / _totalStaked);
```

Description

biakia : In contract `ARBRewarder`, the function `_updatePool` will calculate the `accARBPerShare` based on the following code:

```
        uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
        uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;

        pool.accARBPerShare = pool.accARBPerShare + ((ARBReward * 1e12) / _totalStaked);
        pool.lastRewardTimestamp = block.timestamp;
```

it is possible to encounter a rounding-down issue when calculating the `accARBPerShare`. Let's say the `multiplier` is 1 second and `ARBPerSec` is 1e14 arb(0.0001 arb per second). The `pool.allocPoint/totalAllocPoint` is 1%.

The `ARBReward` will be `1 * 1e14 * 1 / 100 = 1e12`. The `_totalStaked` is the amount of the token staked in `MasterPenpie`. Let's say the staked token has a decimal of 18, so when the `_totalStaked` is greater than `1e6 * 1e18`, for example `2e6 * 1e18`, the formula `((ARBReward * 1e12) / _totalStaked)` will round down to 0 (1e12 * 1e12 / 2e24 = 0).In this case, the arb rewards will not be added to `pool.accARBPerShare ` due to rounding-down issue.

What's more, when the staked token is a meme token, the `_totalStaked` will be much larger. One user can easily stake billions of meme tokens and the rounding-down is more likely to happen.

rajatbeladiya :

```
uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;

        pool.accARBPerShare = pool.accARBPerShare + ((ARBReward * 1e12) / _totalStaked);
```

Because of Division before Multiplication, the calculation `((ARBReward * 1e12) / _totalStaked)` may lead to precision loss when updating the `accARBPerShare` in the `pool` struct. `ARBReward` divided by `totalAllocPoint` first and multiplication with `1e12` after. This can result in inaccuracies in the distribution of rewards and potential loss of precision when dealing with fractional values.

Recommendation

biakia : Consider using a larger amplification factor, for example:

```
        uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
        uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;

        pool.accARBPerShare = pool.accARBPerShare + ((ARBReward * 1e24) / _totalStaked);
        pool.lastRewardTimestamp = block.timestamp;
```

rajatbeladiya : Remove `ARBReward` calculation and update `pool.accARBPerShare`

```
pool.accARBPerShare = pool.accARBPerShare + ((multiplier * ARBPerSec * pool.allocPoint * 1e12) / (_t
otalStaked * totalAllocPoint));
```

Client Response

Fixed. Changed the precision amplification factor to 1e24.

```
uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint * 1e24) / totalAllocPoint;
pool.accARBPerShare = pool.accARBPerShare + (ARBReward / _totalStaked);
```

MPA-2:Unclaimed Arb rewards will not be sent to the user

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L265-L293

```
function _calculateAndSendARB(
        address _user,
        address[] calldata _stakingTokens,
        address _receiver,
        address _masterChef
    ) internal returns(uint256 totalARBSent){

        uint256 length = _stakingTokens.length;
        uint256 totalARBReward = 0;

        for (uint256 i = 0; i < length; ++i) {
            address stakingToken = _stakingTokens[i];
            UserInfo storage user = userInfo[stakingToken][_user];

            uint256 totalStaked = _getTotalStaked(stakingToken, _masterChef);
            uint256 userStaked = _getUserStaked(stakingToken, _user, _masterChef);
            _updatePool(stakingToken, totalStaked);

            uint256 claimableARB = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12 - user.rewardDebt;
            totalARBReward += claimableARB;
            user.unClaimedARB = 0;
            user.rewardDebt = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12;
        }

        if (totalARBReward > 0) {
            _sendARB(_user, _receiver, totalARBReward);
        }
        totalARBSent = totalARBReward;
    }
```

Description

biakia : In contract `MasterMagpie`, the function `_deposit` will call `ARBRewarder.harvestARB()` to claim arb rewards:

```
 if (user.amount > 0) {
            _harvestMGP(_stakingToken, _account);

            if(isARBIncentivePool[_stakingToken])
                IARBRewarder(ARBRewarder).harvestARB(_stakingToken, _account);
        }
```

In contract `ARBRewarder`, the function `harvestARB` is used to harvest ARB for an account. It will only update the `unClaimedARB` of the user record:

```
userInfo[_stakingToken][_account].unClaimedARB += pending;
```

After the call of `harvestARB`, the function `_deposit` will call `ARBRewarder.updateRewardDebt` to update the debt:

```
if(isARBIncentivePool[_stakingToken])
            IARBRewarder(ARBRewarder).updateRewardDebt(_account, address(0), _stakingToken);
```

In contract `ARBRewarder`, the function `updateRewardDebt` will update the user's debt to the latest value:

```
function updateRewardDebt(
        address _from,
        address _to,
        address _stakingToken
    ) external _onlyMasterChef {

        address masterChef = msg.sender;
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];

        if (_from != address(0)) {
            UserInfo storage from = userInfo[_stakingToken][_from];
            uint256 fromAmount = _getUserStaked(_stakingToken, _from, masterChef);
            from.rewardDebt = (fromAmount * pool.accARBPerShare) / 1e12;
        }
        if (_to != address(0)) {
            UserInfo storage to = userInfo[_stakingToken][_to];
            uint256 toAmount = _getUserStaked(_stakingToken, _to, masterChef);
            to.rewardDebt = (toAmount * pool.accARBPerShare) / 1e12;
        }
    }
```

All unclaimed arb rewards now are recorded in the variable `unClaimedARB`. When the user claims these arb rewards, the function `_multiClaim` will be called and it will call `ARBRewarder.sendARBRewards()`:

```
 function _sendARBRewards(address _user, address[] calldata _stakingTokens, address _receiver) inter
nal {

        uint256 arbRewardPoolCount;
        uint256 length = _stakingTokens.length;

        for(uint256 i = 0; i < length; i++){
            if(isARBIncentivePool[_stakingTokens[i]])
                arbRewardPoolCount++;
        }

        address[] memory arbRewardPools = new address[](arbRewardPoolCount);
        uint256 index = 0;

        for (uint256 i = 0; i < length; ++i) {
            address _stakingToken = _stakingTokens[i];
            if(isARBIncentivePool[_stakingToken])
                arbRewardPools[index++] = _stakingToken;
        }
        IARBRewarder(ARBRewarder).sendARBRewards(_user, arbRewardPools, _receiver);
    }
```

In contract `ARBRewarder`, the function `sendARBRewards` will call function `_calculateAndSendARB`. Here we can see that in function `_calculateAndSendARB`, the `unClaimedARB` will not be added to `totalARBReward` and it will be reset as 0:

```
uint256 claimableARB = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12 - user.rew
ardDebt;
            totalARBReward += claimableARB;
            user.unClaimedARB = 0;
            user.rewardDebt = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12;
```

At last, the user will lose these unclaimed Arb rewards.

Recommendation

biakia : Consider adding `unClaimedARB` to `totalARBReward`:

```
uint256 claimableARB = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12 - user.rew
ardDebt;
            totalARBReward += claimableARB;
            totalARBReward += user.unClaimedARB;
            user.unClaimedARB = 0;
            user.rewardDebt = (userStaked * tokenToPoolInfo[stakingToken].accARBPerShare) / 1e12;
```

Client Response

Fixed.This change was included in the latest version in the commit 435e21f61bbfdc64ce4248c8bc0f47ca5b1b451b

MPA-3:The `depositMPendleSVFor` and `depositVlPenpieFor` functions did not update ARBRewarder's debt, causing the user to get more arb rewards than expected

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/penpie-contracts/contracts/rewards/MasterPenpie.sol#L606-L670

```
function _deposit(
        address _stakingToken,
        address _from,
        address _for,
        uint256 _amount,
        bool _isLock
    ) internal {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
        UserInfo storage user = userInfo[_stakingToken][_for];

        updatePool(_stakingToken);
        _harvestRewards(_stakingToken, _for);

        user.amount = user.amount + _amount;
        if (!_isLock) {
            user.available = user.available + _amount;
            IERC20(pool.stakingToken).safeTransferFrom(
                address(_from),
                address(this),
                _amount
            );
        }
        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;

        if (_amount > 0) {
            pool.totalStaked += _amount;
            if (!_isLock)
                emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
            else emit DepositNotAvailable(_for, _stakingToken, _amount);
        }
    }

    /// @notice internal function to deal with withdraw staking token
    function _withdraw(
        address _stakingToken,
        address _account,
        uint256 _amount,
        bool _isLock
    ) internal {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
        UserInfo storage user = userInfo[_stakingToken][_account];

        if (!_isLock && user.available < _amount)
            revert WithdrawAmountExceedsStaked();
        else if (user.amount < _amount && _isLock)
            revert UnlockAmountExceedsLocked();

        updatePool(_stakingToken);
        _harvestPenpie(_stakingToken, _account);
        _harvestBaseRewarder(_stakingToken, _account);

        user.amount = user.amount - _amount;
        if (!_isLock) {
            user.available = user.available - _amount;
            IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
                address(msg.sender),
                _amount
            );
        }
        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;

        pool.totalStaked -= _amount;

        emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
    }
```

Description

biakia : Both `depositMPendleSVFor` and `depositVlPenpieFor` functions will call the function `_deposit`:

```
function _deposit(
        address _stakingToken,
        address _from,
        address _for,
        uint256 _amount,
        bool _isLock
    ) internal {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
        UserInfo storage user = userInfo[_stakingToken][_for];

        updatePool(_stakingToken);
        _harvestRewards(_stakingToken, _for);

        user.amount = user.amount + _amount;
        if (!_isLock) {
            user.available = user.available + _amount;
            IERC20(pool.stakingToken).safeTransferFrom(
                address(_from),
                address(this),
                _amount
            );
        }
        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;

        if (_amount > 0) {
            pool.totalStaked += _amount;
            if (!_isLock)
                emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
            else emit DepositNotAvailable(_for, _stakingToken, _amount);
        }
    }
```

In `_deposit` function, it will call `_harvestRewards` function:

```
function _harvestRewards(address _stakingToken, address _account) internal {
        if (userInfo[_stakingToken][_account].amount > 0) {
            _harvestPenpie(_stakingToken, _account);

            if(isARBIncentivePool[_stakingToken])
                IARBRewarder(ARBRewarder).harvestARB(_stakingToken, _account);
        }
        _harvestBaseRewarder(_stakingToken, _account);
    }
```

In `_harvestRewards` function, it will call `IARBRewarder(ARBRewarder).harvestARB` to harvest the reward:

```
function harvestARB(address _stakingToken, address _account) external _onlyMasterChef {

        address masterChef = msg.sender;
        uint256 userStaked = _getUserStaked(_stakingToken, _account, masterChef);

        uint256 pending = (userStaked * tokenToPoolInfo[_stakingToken].accARBPerShare) /
            1e12 - userInfo[_stakingToken][_account].rewardDebt;

        userInfo[_stakingToken][_account].unClaimedARB += pending;
    }
```

In contract `ARBRewarder`, the function `harvestARB` will add the pending rewards to `unClaimedARB` but not update the user's debt. The debt will only be updated in function `updateRewardDebt`. After `_deposit` function calls `_harvestRewards` function, it does not call the function `ARBRewarder.updateRewardDebt`, which means the user's debt is still the same as the original debt. Since the arb reward is calculated by the user's debt, if the user's debt is not updated, when the functions `depositMPendleSVFor` and `depositVlPenpieFor` are called again, the user can get more rewards than expected.

What's more, the function `_withdraw` neither harvests arb rewards nor updates user's debt. When the user unstakes all tokens, he will not harvest arb rewards and he has to deposit once more to harvest arb rewards.

Recommendation

biakia : Consider updating the debt in `_deposit` function:

```
function _deposit(
        address _stakingToken,
        address _from,
        address _for,
        uint256 _amount,
        bool _isLock
    ) internal {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
        UserInfo storage user = userInfo[_stakingToken][_for];

        updatePool(_stakingToken);
        _harvestRewards(_stakingToken, _for);

        user.amount = user.amount + _amount;
        if (!_isLock) {
            user.available = user.available + _amount;
            IERC20(pool.stakingToken).safeTransferFrom(
                address(_from),
                address(this),
                _amount
            );
        }
        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
        if(isARBIncentivePool[_stakingToken])
            IARBRewarder(ARBRewarder).updateRewardDebt(_for, address(0), _stakingToken);
        if (_amount > 0) {
            pool.totalStaked += _amount;
            if (!_isLock)
                emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
            else emit DepositNotAvailable(_for, _stakingToken, _amount);
        }
    }
```

Consider adding the same logic in `_withdraw` function:

```
  function _withdraw(
          address _stakingToken,
          address _account,
          uint256 _amount,
          bool _isLock
      ) internal {
          PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
          UserInfo storage user = userInfo[_stakingToken][_account];

          if (!_isLock && user.available < _amount)
              revert WithdrawAmountExceedsStaked();
          else if (user.amount < _amount && _isLock)
              revert UnlockAmountExceedsLocked();
          if(isARBIncentivePool[_stakingToken])
                  IARBRewarder(ARBRewarder).harvestARB(_stakingToken, _account);
          updatePool(_stakingToken);
          _harvestPenpie(_stakingToken, _account);
          _harvestBaseRewarder(_stakingToken, _account);

          user.amount = user.amount - _amount;
          if (!_isLock) {
              user.available = user.available - _amount;
              IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
                  address(msg.sender),
                  _amount
              );
          }
          user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;

          pool.totalStaked -= _amount;
          if(isARBIncentivePool[_stakingToken])
              IARBRewarder(ARBRewarder).updateRewardDebt(_account, address(0), _stakingToken);
          emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
      }
```

Client Response

Fixed.Added the corresponding logic in _withdraw & _deposit in commit 4a43404e0e9a3f76f6a77de4f30073a4c780175d in Penpie and ed0524a2b5d5af286b8d1979be32445aa174aa2d in Radpie

MPA-4:The utilization of the function `_getUserStaked()` in the contract is vulnerable to sandwich attack.

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	n16h7m4r3

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L187
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L209
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L214
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L238
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L280

```
187:uint256 userStaked = _getUserStaked(_stakingToken, _account, masterChef);

209:uint256 fromAmount = _getUserStaked(_stakingToken, _from, masterChef);

214:uint256 toAmount = _getUserStaked(_stakingToken, _to, masterChef);

238:uint256 userStaked = _getUserStaked(_stakingToken, _user, _masterChef);

280:uint256 userStaked = _getUserStaked(stakingToken, _user, _masterChef);
```

Description

n16h7m4r3 : The function `_getUserStaked()` fetches the amount of tokens staked by an wallet to compute ARB rewards. The function can be sandwiched by a wallet using the functions `deposit()` and `withdraw()` in the `Master Penpie` contract to obtain higher rewards.

Recommendation

n16h7m4r3 : Business logic issue, can be consider having a lock period for the tokens or computing rewards based on time staked in the pool.

Client Response

Acknowledged.If a user tries to sandwitch a ARB claim transaction between a deposit and a withdraw, the deposit itself will update the accARBPerShare and the user's ARB rewards until that moment get stored in userInfo.unclaimedARB and the userInfo.rewardDebt also gets updated accordingly. Similar is the case with withdraw. Any change in the totalStaked or userStaked in masterPenpie also updates the accARBPerShare and other variables correspondigly.

The other functions where _getUserStaked is used are called whenever a deposit or withdraw is called.

MPA-5: `getTokenPrice` will return wrong price when chainlink usd price feed's `decimal != 8`

Category	Severity	Client Response	Contributor
Logical	**Medium**	**Acknowledged**	**biakia**

Code Reference

- code/magpie_contracts/contracts/MagpieReader.sol#L459-L468
- code/magpie_contracts/contracts/MagpieReaderArb.sol#L482-L491
- code/magpie_contracts/contracts/MagpieReaderEth.sol#L482-L491

```
459:else if (tokenRouter.routerType == ChainlinkType) {
460:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chai
nlink);
461:              (
462:                /* uint80 roundID */,
463:                int256 price,
464:                /*uint startedAt*/,
465:                /*uint timeStamp*/,
466:                /*uint80 answeredInRound*/
467:            ) = aggregatorV3Interface.latestRoundData();
468:            amountOut = uint256(price * 1e18 / 1e8);

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chai
nlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
487:                /*uint startedAt*/,
488:                /*uint timeStamp*/,
489:                /*uint80 answeredInRound*/
490:            ) = aggregatorV3Interface.latestRoundData();
491:            amountOut = uint256(price * 1e18 / 1e8);

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chai
nlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
487:                /*uint startedAt*/,
488:                /*uint timeStamp*/,
489:                /*uint80 answeredInRound*/
490:            ) = aggregatorV3Interface.latestRoundData();
491:            amountOut = uint256(price * 1e18 / 1e8);
```

Description

biakia : In contract `MagpieReader`, `MagpieReaderArb` and `MagpieReaderEth`, the function `getTokenPrice` will use chainlink to fetch token price. It assumes that the price feed's decimal is always equal to 8:

```
amountOut = uint256(price * 1e18 / 1e8);
```

However, there are tokens with USD price feed's decimal != 8, for example, the AMPL/USD token feed(https://etherscan.io/address/0xe20ca8d7546932360e37e9d72c1a47334af57706). When the price feed's decimal != 8, the function `getTokenPrice` will return an incorrect price.

Recommendation

biakia : Consider adding a check on the feed's decimal :

```
AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
            require(aggregatorV3Interface.decimals() == 8,"invalid decimal");
              (
                /* uint80 roundID */,
                int256 price,
                /*uint startedAt*/,
                /*uint timeStamp*/,
                /*uint80 answeredInRound*/
            ) = aggregatorV3Interface.latestRoundData();
            amountOut = uint256(price * 1e18 / 1e8);
```

Client Response

Acknowledged, magpieReader is only used by front end to render data, so not of much concern

MPA-6:Chainlink's latestRoundData might return stale results

Category	Severity	Client Response	Contributor
Oracle Manipulation	**Medium**	**Acknowledged**	**biakia, rajatbeladiya**

Code Reference

- code/magpie_contracts/contracts/MagpieReader.sol#L459-L469
- code/magpie_contracts/contracts/MagpieReader.sol#L460-L467
- code/magpie_contracts/contracts/MagpieReaderArb.sol#L482-L492
- code/magpie_contracts/contracts/MagpieReaderEth.sol#L482-L492

```
459:else if (tokenRouter.routerType == ChainlinkType) {
460:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
461:              (
462:                /* uint80 roundID */,
463:                int256 price,
464:                /*uint startedAt*/,
465:                /*uint timeStamp*/,
466:                /*uint80 answeredInRound*/
467:            ) = aggregatorV3Interface.latestRoundData();
468:            amountOut = uint256(price * 1e18 / 1e8);
469:           }

460:AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
461:              (
462:                /* uint80 roundID */,
463:                int256 price,
464:                /*uint startedAt*/,
465:                /*uint timeStamp*/,
466:                /*uint80 answeredInRound*/
467:            ) = aggregatorV3Interface.latestRoundData();

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
487:                /*uint startedAt*/,
488:                /*uint timeStamp*/,
489:                /*uint80 answeredInRound*/
490:            ) = aggregatorV3Interface.latestRoundData();
491:            amountOut = uint256(price * 1e18 / 1e8);
492:           } else if (tokenRouter.routerType == UniswapV3RouterType) {

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
                /*uint startedAt*/,
                /*uint timeStamp*/,
                /*uint80 answeredInRound*/
            ) = aggregatorV3Interface.latestRoundData();
            amountOut = uint256(price * 1e18 / 1e8);
           } else if (tokenRouter.routerType == UniswapV3RouterType) {
```

Description

biakia : In contract `MagpieReader`, the function `getTokenPrice` will use `aggregatorV3Interface.latestRoundData()` to fetch latest price:

```
AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
              (
                /* uint80 roundID */,
                int256 price,
                /*uint startedAt*/,
                /*uint timeStamp*/,
                /*uint80 answeredInRound*/
            ) = aggregatorV3Interface.latestRoundData();
            amountOut = uint256(price * 1e18 / 1e8);
```

The issue here is that there is no check for the last updated time for the price. So we would not know if the priced returned exceeded the timeout. It may return an expired price and incur unexpected side effects.

The same issue exits in contract `MagpieReaderArb` and `MagpieReaderEth`.

rajatbeladiya : here Magpie using Chainlink's latestRoundData API, but there is no check if the return value indicates stale data. This could lead to stale prices according to the Chainlink documentation:

https://docs.chain.link/docs/historical-price-data/#historical-rounds

Recommendation

biakia : Consider adding a check to see when the price was last updated and revert if the price is older than a certain time period:

```
AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
              (
                /* uint80 roundID */,
                int256 price,
                /*uint startedAt*/,
                uint timeStamp,
                /*uint80 answeredInRound*/
            ) = aggregatorV3Interface.latestRoundData();
            require(price > 0, "Error: Invalid price");
            require(timeStamp > block.timestamp - MAX_TIME_DELAY, "Error: Invalid updated time");
            amountOut = uint256(price * 1e18 / 1e8);
```

rajatbeladiya : Add check for stale data

```
AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
              (
                uint80 roundID,
                int256 price,
                uint timeStamp,
                uint80 answeredInRound,
            ) = aggregatorV3Interface.latestRoundData();
require(answeredInRound >= roundId, 'stale price');
require(timestamp != 0, 'round not complete');
```

Client Response

Acknowledged, MagpieReader is only for frontend to render the data so not of much concern

MPA-7:potential DOS attack in `MasterMagpie::_sendARBRewards` function

Category	Severity	Client Response	Contributor
DOS	Low	Acknowledged	ginlee

Code Reference

- code/radpie_contracts/contracts/rewards/MasterRadpie.sol#L517-L536
- code/magpie_contracts/contracts/rewards/MasterMagpie.sol#L748-L767

```
function _sendARBRewards(address _user, address[] calldata _stakingTokens, address _receiver) internal {

        uint256 arbRewardPoolCount;
        uint256 length = _stakingTokens.length;

        for(uint256 i = 0; i < length; i++){
            if(isARBIncentivePool[_stakingTokens[i]])
                arbRewardPoolCount++;
        }

        address[] memory arbRewardPools = new address[](arbRewardPoolCount);
        uint256 index = 0;

        for (uint256 i = 0; i < length; ++i) {
            address _stakingToken = _stakingTokens[i];
            if(isARBIncentivePool[_stakingToken])
                arbRewardPools[index++] = _stakingToken;
        }
        IARBRewarder(ARBRewarder).sendARBRewards(_user, arbRewardPools, _receiver);
    }

function _sendARBRewards(address _user, address[] calldata _stakingTokens, address _receiver) internal {

        uint256 arbRewardPoolCount;
        uint256 length = _stakingTokens.length;

        for(uint256 i = 0; i < length; i++){
            if(isARBIncentivePool[_stakingTokens[i]])
                arbRewardPoolCount++;
        }

        address[] memory arbRewardPools = new address[](arbRewardPoolCount);
        uint256 index = 0;

        for (uint256 i = 0; i < length; ++i) {
            address _stakingToken = _stakingTokens[i];
            if(isARBIncentivePool[_stakingToken])
                arbRewardPools[index++] = _stakingToken;
        }
        IARBRewarder(ARBRewarder).sendARBRewards(_user, arbRewardPools, _receiver);
    }
```

Description

ginlee :

```
    for (uint256 i = 0; i < length; ++i) {
            address _stakingToken = _stakingTokens[i];
            if(isARBIncentivePool[_stakingToken])
                arbRewardPools[index++] = _stakingToken;
        }
```

The size of the _stakingTokens array is based on user input, which means that users can affect the resource consumption of the function. If a malicious user intentionally passes in a very large array, it could increase the complexity of the function execution

Recommendation

ginlee : Limit array size: You can add logic within the function to restrict the maximum size of the user-input array, preventing potential abuse.

Client Response

Acknowledged.This particular logic has been changed but it still iterates over the entire _stakingTokens array but that's necessary.

MPA-8:Missing check Arbitrum sequencer status when fetching prices from Chainlink feed

Category	Severity	Client Response	Contributor
Oracle Manipulation	**Low**	**Acknowledged**	**biakia**

Code Reference

- code/magpie_contracts/contracts/MagpieReader.sol#L459-L469
- code/magpie_contracts/contracts/MagpieReaderArb.sol#L482-L492
- code/magpie_contracts/contracts/MagpieReaderEth.sol#L482-L492

```
459:else if (tokenRouter.routerType == ChainlinkType) {
460:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
461:              (
462:                /* uint80 roundID */,
463:                int256 price,
464:                /*uint startedAt*/,
465:                /*uint timeStamp*/,
466:                /*uint80 answeredInRound*/
467:            ) = aggregatorV3Interface.latestRoundData();
468:            amountOut = uint256(price * 1e18 / 1e8);
469:           }

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
487:                /*uint startedAt*/,
488:                /*uint timeStamp*/,
489:                /*uint80 answeredInRound*/
490:            ) = aggregatorV3Interface.latestRoundData();
491:            amountOut = uint256(price * 1e18 / 1e8);
492:           } else if (tokenRouter.routerType == UniswapV3RouterType) {

482:else if (tokenRouter.routerType == ChainlinkType) {
483:            AggregatorV3Interface aggregatorV3Interface = AggregatorV3Interface(tokenRouter.chainlink);
484:              (
485:                /* uint80 roundID */,
486:                int256 price,
487:                /*uint startedAt*/,
488:                /*uint timeStamp*/,
489:                /*uint80 answeredInRound*/
490:            ) = aggregatorV3Interface.latestRoundData();
491:            amountOut = uint256(price * 1e18 / 1e8);
492:           } else if (tokenRouter.routerType == UniswapV3RouterType) {
```

Description

biakia : When using Chainlink oracles on optimistic rollups, there should be a validation on the L2 sequencer is up and active when consuming price feeds. Chainlink recommends that all optimistic L2 oracles consult the Sequencer Uptime Feed to ensure the sequencer is live. https://docs.chain.link/data-feeds/l2-sequencer-feeds

Recommendation

biakia : Consider ensuring that the sequencer is live before getting latest price from chainlink.

Client Response

Acknowledged.Not of much concern as this data is only used for UI rendering.

MPA-9:Not check whether the pool exists when updating `allocPoint`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L403-L423

```
403:function updatePoolsAlloc(
404:        address[] calldata _stakingTokens,
405:        uint256[] calldata _allocPoints
406:    ) external onlyOwner {
407:
408:        if (_stakingTokens.length != _allocPoints.length)
409:            revert LengthMismatch();
410:        massUpdatePools();
411:        for (uint256 i = 0; i < _stakingTokens.length; i++) {
412:
413:            uint256 oldAllocPoint = tokenToPoolInfo[_stakingTokens[i]].allocPoint;
414:            totalAllocPoint = totalAllocPoint - oldAllocPoint + _allocPoints[i];
415:            tokenToPoolInfo[_stakingTokens[i]].allocPoint = _allocPoints[i];
416:
417:            emit UpdatePoolAlloc(
418:                _stakingTokens[i],
419:                oldAllocPoint,
420:                _allocPoints[i]
421:            );
422:        }
423:    }
```

Description

biakia : In contract `ARBRewarder`, the function `updatePoolsAlloc` is used to update `allocPoint` of the pool:

```
function updatePoolsAlloc(
        address[] calldata _stakingTokens,
        uint256[] calldata _allocPoints
    ) external onlyOwner {

        if (_stakingTokens.length != _allocPoints.length)
            revert LengthMismatch();
        massUpdatePools();
        for (uint256 i = 0; i < _stakingTokens.length; i++) {

            uint256 oldAllocPoint = tokenToPoolInfo[_stakingTokens[i]].allocPoint;
            totalAllocPoint = totalAllocPoint - oldAllocPoint + _allocPoints[i];
            tokenToPoolInfo[_stakingTokens[i]].allocPoint = _allocPoints[i];

            emit UpdatePoolAlloc(
                _stakingTokens[i],
                oldAllocPoint,
                _allocPoints[i]
            );
        }
    }
```

However, there is no check on whether `tokenToPoolInfo[_stakingTokens[i]]` exists. If `tokenToPoolInfo[_stakingTokens[i]]` not exists, some of the rewards will be left in the contract because some `allocPoint` has been allocated to an invalid pool.

Recommendation

biakia : Consider adding a check on whether the pool exists:

```
function updatePoolsAlloc(
        address[] calldata _stakingTokens,
        uint256[] calldata _allocPoints
    ) external onlyOwner {

        if (_stakingTokens.length != _allocPoints.length)
            revert LengthMismatch();
        massUpdatePools();
        for (uint256 i = 0; i < _stakingTokens.length; i++) {
            require(tokenToPoolInfo[stakingTokens[i]].isActive,"invalid pool");
            uint256 oldAllocPoint = tokenToPoolInfo[_stakingTokens[i]].allocPoint;
            totalAllocPoint = totalAllocPoint - oldAllocPoint + _allocPoints[i];
            tokenToPoolInfo[_stakingTokens[i]].allocPoint = _allocPoints[i];

            emit UpdatePoolAlloc(
                _stakingTokens[i],
                oldAllocPoint,
                _allocPoints[i]
            );
        }
    }
```

Client Response

Fixed.Added the check

```
if(!tokenToPoolInfo[_stakingTokens[i]].isActive)
                revert OnlyActivePool();
```

MPA-10:There is no function to initialize `MagpieReaderArb`

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	biakia

Code Reference

- code/magpie_contracts/contracts/MagpieReaderArb.sol#L286-L309

```
286:/* ============ Constructor ============ */
287:
288:    // function __MagpieReader_init(
289:    //      IMasterMagpieReader _masterMagpie,
290:    //      IWombatBribeManagerReader _wombatBribeManager,
291:    //      IPancakeRouter02Reader _pancakeRouter02,
292:    //      IWombatRouterReader _wombatRouter,
293:    //      address _mWomSV
294:    //      )
295:    //      public
296:    //      initializer
297:    // {
298:    //      __Ownable_init();
299:    //      masterMagpie = _masterMagpie;
300:    //      wombatBribeManager = _wombatBribeManager;
301:    //      pancakeRouter02 = _pancakeRouter02;
302:    //      wombatRouter = _wombatRouter;
303:    //      mWomSV = _mWomSV;
304:    //      voter = IWombatVoter(wombatBribeManager.voter());
305:    //      wombatStaking = IWombatStakingReader(wombatBribeManager.wombatStaking());
306:    //      masterWombatV3 = IMasterWombatV3Reader(wombatStaking.masterWombat());
307:    // }
308:
309:    /* ============ External Getters ============ */
```

Description

biakia : In contract `MagpieReaderArb`, the function `__MagpieReader_init` is commented:

```
// function __MagpieReader_init(
    //     IMasterMagpieReader _masterMagpie,
    //     IWombatBribeManagerReader _wombatBribeManager,
    //     IPancakeRouter02Reader _pancakeRouter02,
    //     IWombatRouterReader _wombatRouter,
    //     address _mWomSV
    //     )
    //     public
    //     initializer
    // {
    //     __Ownable_init();
    //     masterMagpie = _masterMagpie;
    //     wombatBribeManager = _wombatBribeManager;
    //     pancakeRouter02 = _pancakeRouter02;
    //     wombatRouter = _wombatRouter;
    //     mWomSV = _mWomSV;
    //     voter = IWombatVoter(wombatBribeManager.voter());
    //     wombatStaking = IWombatStakingReader(wombatBribeManager.wombatStaking());
    //     masterWombatV3 = IMasterWombatV3Reader(wombatStaking.masterWombat());
    // }
```

Important parameters like `masterMagpie`, `pancakeRouter02` will be `address(0)` as default, and functions like `getMagpieInfo`, `getTokenPrice` will not work properly. What's more, the function `__Ownable_init()` will not be called when the contract is deployed, which means the `owner` will be `address(0)` as default.

Recommendation

biakia : Consider uncommenting the `__MagpieReader_init` function.

Client Response

Acknowledged

MPA-11:Use `disableInitializers` to prevent any future reinitialization

Category	Severity	Client Response	Contributor
Code Style	**Low**	**Acknowledged**	**biakia**

Code Reference

- code/radpie_contracts/contracts/rewards/MasterRadpie.sol#L174-L187
- code/magpie_contracts/contracts/rewards/MasterMagpie.sol#L209-L220
- code/magpie_contracts/contracts/MagpieReader.sol#L282-L299
- code/magpie_contracts/contracts/MagpieReaderEth.sol#L289-L308

```
function __MasterRadpie_init(
        address _radpie,
        uint256 _radpiePerSec,
        uint256 _startTimestamp
    ) public initializer {
        __Ownable_init();
        __ReentrancyGuard_init();
        __Pausable_init();
        radpie = IERC20(_radpie);
        radpiePerSec = _radpiePerSec;
        startTimestamp = _startTimestamp;
        totalAllocPoint = 0;
        PoolManagers[owner()] = true;
    }

function __MasterMagpie_init(
        address _mgp,
        uint256 _mgpPerSec,
        uint256 _startTimestamp
    ) public initializer {
        __Ownable_init();
        mgp = MGP(_mgp);
        mgpPerSec = _mgpPerSec;
        startTimestamp = _startTimestamp;
        totalAllocPoint = 0;
        PoolManagers[owner()] = true;
    }

function __MagpieReader_init(
        IMasterMagpieReader _masterMagpie,
        IWombatBribeManagerReader _wombatBribeManager,
        IPancakeRouter02Reader _pancakeRouter02,
        IWombatRouterReader _wombatRouter
        )
        public
        initializer
    {
        __Ownable_init();
        masterMagpie = _masterMagpie;
        wombatBribeManager = _wombatBribeManager;
        pancakeRouter02 = _pancakeRouter02;
        wombatRouter = _wombatRouter;
        voter = IWombatVoter(wombatBribeManager.voter());
        wombatStaking = IWombatStakingReader(wombatBribeManager.wombatStaking());
        masterWombatV3 = IMasterWombatV3Reader(wombatStaking.masterWombat());
    }

function __MagpieReader_init(
        IMasterMagpieReader _masterMagpie,
        IWombatBribeManagerReader _wombatBribeManager,
        IPancakeRouter02Reader _pancakeRouter02,
        IWombatRouterReader _wombatRouter,
        address _mWomSV
        )
        public
        initializer
    {
        __Ownable_init();
        masterMagpie = _masterMagpie;
        wombatBribeManager = _wombatBribeManager;
        pancakeRouter02 = _pancakeRouter02;
        wombatRouter = _wombatRouter;
        mWomSV = _mWomSV;
        voter = IWombatVoter(wombatBribeManager.voter());
        wombatStaking = IWombatStakingReader(wombatBribeManager.wombatStaking());
        masterWombatV3 = IMasterWombatV3Reader(wombatStaking.masterWombat());
    }
```

Description

biakia : The `MasterRadpie`, `MasterMagpie`, `MagpieReader`, `MagpieReaderArb` and `MagpieReaderEth` are upgradeable contracts and can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any value in the proxy contract. However, taking ownership of implementation contracts can open other attack vectors, like social engineer or phishing attack. See docs: https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--

Recommendation

biakia : Consider using `disableInitializers` in these contracts:

```
constructor() {
        _disableInitializers();
}
```

Client Response

Acknowledged

MPA-12:The function `pendingARB` maybe use an incorrect `_masterChef` address

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L159-L161
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L228-L248

```
159:function pendingARB( address _stakingToken, address _user ) public view returns (uint256 userPendingARB) {
160:        userPendingARB = _calARBReward(_stakingToken, _user, msg.sender);
161:    }

228:function _calARBReward(
229:        address _stakingToken,
230:        address _user,
231:        address _masterChef
232:    ) internal view returns (uint256 userPendingARB) {
233:        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
234:        UserInfo storage user = userInfo[_stakingToken][_user];
235:        uint256 accARBPerShare = pool.accARBPerShare;
236:
237:        uint256 totalStaked = _getTotalStaked(_stakingToken, _masterChef);
238:        uint256 userStaked = _getUserStaked(_stakingToken, _user, _masterChef);
239:
240:        if (block.timestamp > pool.lastRewardTimestamp && totalStaked != 0) {
241:            uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
242:            uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;
243:            accARBPerShare = accARBPerShare + (ARBReward * 1e12) / totalStaked;
244:        }
245:
246:        userPendingARB = (userStaked * accARBPerShare) / 1e12 - user.rewardDebt;
247:        userPendingARB += user.unClaimedARB;
248:    }
```

Description

biakia : In contract `ARBRewarder`, the function `pendingARB` is used for frontend users to see pending reward tokens. It will use `msg.sender` as the param `_masterChef`:

```
  function pendingARB( address _stakingToken, address _user ) public view returns (uint256 userPending
  ARB) {
          userPendingARB = _calARBReward(_stakingToken, _user, msg.sender);
      }
```

In function `_calARBReward`, the `_masterChef` is used to get `totalStaked` and `userStaked`:

```
  function _calARBReward(
          address _stakingToken,
          address _user,
          address _masterChef
      ) internal view returns (uint256 userPendingARB) {
          PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
          UserInfo storage user = userInfo[_stakingToken][_user];
          uint256 accARBPerShare = pool.accARBPerShare;

          uint256 totalStaked = _getTotalStaked(_stakingToken, _masterChef);
          uint256 userStaked = _getUserStaked(_stakingToken, _user, _masterChef);
          if (block.timestamp > pool.lastRewardTimestamp && totalStaked != 0) {
              uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
              uint256 ARBReward = (multiplier * ARBPerSec * pool.allocPoint) / totalAllocPoint;
              accARBPerShare = accARBPerShare + (ARBReward * 1e12) / totalStaked;
          }

          userPendingARB = (userStaked * accARBPerShare) / 1e12 - user.rewardDebt;
          userPendingARB += user.unClaimedARB;
      }

   function _getTotalStaked(address _stakingToken, address masterChef) internal view returns (uint256
  totalStaked){
          (,,totalStaked,) = IMasterPenpie(masterChef).getPoolInfo(_stakingToken);
      }
  function _getUserStaked(address _stakingToken, address _user, address masterChef) internal view retu
  rns (uint256 userStakedAmount){
          (userStakedAmount,) = IMasterPenpie(masterChef).stakingInfo(_stakingToken, _user);
      }
```

If the `msg.sender` is a EOA address, the `totalStaked` and `userStaked` will be 0. The formula `(userStaked * accARBPerShare) / 1e12 - user.rewardDebt` will be `0 * accARBPerShare/1e12 - user.rewardDebt`. Since `user.rewardDebt` is greater than 0, the function will revert due to underflow error.

Recommendation

biakia : Consider using the `masterChef` in `PoolInfo`:

```
  function pendingARB( address _stakingToken, address _user ) public view returns (uint256 userPending
  ARB) {
          PoolInfo memory pool = tokenToPoolInfo[_stakingToken];
          if(pool.isActive){
          userPendingARB = _calARBReward(_stakingToken, _user, pool.masterChef);
          }
      }
```

Client Response

Fixed.changed pendingARB function to _onlyMasterChef and this function can now be used to get pending rewards:

```
   function calARBReward(
          address _stakingToken,
          address _user,
          address _masterChef
      ) public view returns (uint256 userPendingARB)
```

MPA-13: `massUpdatePools()` is susceptible to DoS with block gas limit

Category	Severity	Client Response	Contributor
Logical	**Low**	**Mitigated**	**thereksfour, rajatbeladiya**

Code Reference

- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L172-L180
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L319-L340
- code/penpie-contracts/contracts/rewards/ARBRewarder.sol#L335

```
function massUpdatePools() public whenNotPaused {

        for (uint256 pid = 0; pid < registeredPools.length; ++pid) {
            address stakingToken = registeredPools[pid];
            address masterChef = tokenToPoolInfo[stakingToken].masterChef;
            uint256 totalStaked = _getTotalStaked(stakingToken, masterChef);
            _updatePool(stakingToken, totalStaked);
        }
    }
```

```
function massUpdatePools() public whenNotPaused {

        for (uint256 pid = 0; pid < registeredPools.length; ++pid) {
            address stakingToken = registeredPools[pid];
            address masterChef = tokenToPoolInfo[stakingToken].masterChef;
            uint256 totalStaked = _getTotalStaked(stakingToken, masterChef);
            _updatePool(stakingToken, totalStaked);
        }
    }
```

```
function _addPool(address _stakingToken, uint256 _allocPoint, address _masterChef) internal {

        if(tokenToPoolInfo[_stakingToken].isActive)
            revert PoolAlreadyAdded();

        totalAllocPoint += _allocPoint;

        tokenToPoolInfo[_stakingToken] = PoolInfo({
            stakingToken: _stakingToken,
            allocPoint: _allocPoint,
            lastRewardTimestamp: block.timestamp > startTimestamp? block.timestamp: startTimestamp,
            accARBPerShare: 0,
            masterChef: _masterChef,
            isActive: true
        });

        registeredPools.push(_stakingToken);

        IMasterPenpie(_masterChef).addPoolsForARBIncentive(_stakingToken);
        emit registeredPool(_stakingToken, _allocPoint);
339:
340:    }

335:registeredPools.push(_stakingToken);
```

Description

thereksfour : massUpdatePools() is a public function and it calls the _updatePool() function for the length of registeredPools.

```
    function massUpdatePools() public whenNotPaused {

        for (uint256 pid = 0; pid < registeredPools.length; ++pid) {
            address stakingToken = registeredPools[pid];
            address masterChef = tokenToPoolInfo[stakingToken].masterChef;
            uint256 totalStaked = _getTotalStaked(stakingToken, masterChef);
            _updatePool(stakingToken, totalStaked);
        }
    }
```

The owner can call addPools to add unlimited registeredPools.

```
    function _addPool(address _stakingToken, uint256 _allocPoint, address _masterChef) internal {

        if(tokenToPoolInfo[_stakingToken].isActive)
            revert PoolAlreadyAdded();

        totalAllocPoint += _allocPoint;

        tokenToPoolInfo[_stakingToken] = PoolInfo({
            stakingToken: _stakingToken,
            allocPoint: _allocPoint,
            lastRewardTimestamp: block.timestamp > startTimestamp? block.timestamp: startTimestamp,
            accARBPerShare: 0,
            masterChef: _masterChef,
            isActive: true
        });

        registeredPools.push(_stakingToken);

        IMasterPenpie(_masterChef).addPoolsForARBIncentive(_stakingToken);
        emit registeredPool(_stakingToken, _allocPoint);

    }
```

Hence, it is an unbounded loop, depending on the length of registeredPools. If registeredPools.length is big enough, block gas limit may be hit.
rajatbeladiya : The loop in the `massUpdatePools` function iterates over the `registeredPools` array. This could potentially lead to a gas denial-of-service (DoS) vulnerability, especially if the array `registeredPools` grows too large.

Recommendation

thereksfour : Consider limiting the length of registeredPools
rajatbeladiya : Consider implement limit to `registeredPools`

Client Response

Mitigated.The addition of new elements in registeredPools array will be managed by magpie owner, massUpdatePools might become much gas heavy when more and more elemets are added in registeredPools but is needed else users might lose rewards. This should be handled by magpie.

MPA-14: Make `AllocationManagers` in MasterMagpie.sol as mapping instead of array as it may cause DOS by unbounded loop

Category	Severity	Client Response	Contributor
DOS	Low	Acknowledged	grep-er

Code Reference

- code/magpie_contracts/contracts/rewards/MasterMagpie.sol#L120
- code/magpie_contracts/contracts/rewards/MasterMagpie.sol#L229

```
120:address[] public AllocationManagers;

229:modifier _onlyWhiteListed() {
```

Description

grep-er : It is used in `_onlyWhiteListed` modifier check if isCallerWhiteListed. For every time this modifier is called it loads all the different AllocationManagers in worst case.

```
    modifier _onlyWhiteListed() {
        bool isCallerWhiteListed = false;
        for (uint i; i < AllocationManagers.length; i++) {
            if (AllocationManagers[i] == msg.sender) {
                isCallerWhiteListed = true;
                break;
            }
        }
```

SLOAD costs 2100 gas to initially access a value during a transaction and costs 100 gas for each subsequent access. But in this case different state variable is called in every loop (`AllocationManagers[i]`) making total cost of gas 2100 * `AllocationManagers.length`

Because of Allocation Manger is array it also makes `removeWhitelistedAllocManager(uint index)` function inefficient.

Recommendation

grep-er : replace array with a mapping of address-->bool to make it easy and efficent and prevent unbounded loops

```
-- address[] public AllocationManagers;

++ mapping(address => bool) public AllocationManagers;
```

Client Response

Acknowledged

MPA-15:No need to use SafeMath in solidity version 0.8+

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	biakia

Code Reference

- code/magpie_contracts/contracts/MagpieReader.sol#L31-L32
- code/magpie_contracts/contracts/MagpieReaderArb.sol#L32-L33
- code/magpie_contracts/contracts/MagpieReaderEth.sol#L33-L34

```
31:using SafeMath for uint256;
32:    using SafeMath for uint128;

32:using SafeMath for uint256;
33:    using SafeMath for uint128;

33:using SafeMath for uint256;
34:    using SafeMath for uint128;
```

Description

biakia : Solidity provides overflow checking for version above 0.8. The contracts ``MagpieReader`, `MagpieReaderArb`a nd`MagpieReaderEth`do not need to import`SafeMath` library for overflow checking, which can save gas.

Recommendation

biakia : Consider removing `SafeMath` library.

Client Response

Fixed.

Disclaimer