EMP Fusion

Security Review Report

February, 2024

Introduction

A security review of the **EMP Fusion** protocol was done by **Reviewer**, with a focus on the security aspects of the application's implementation.

The objective of the assessment was to:

- Identify potential security vulnerabilities within the smart contracts.
- Verify that smart contracts functionality as intended.

Disclaimer

A smart contract security review can never prove the complete absence of vulnerabilities. This review is a time, resource and expertise bound effort to find as many vulnerabilities as possible. However it can not guarantee 100% security of the protocol in any way. Any modifications to the code will require a new security review.

About **Reviewer**

Reviewer is an independent smart contract security researcher who specializes in smart contract audits. He has experience on every part of the blockchain technology stack, including but not limited to protocol design and smart contracts.

About **EMP Fusion**

EMP Fusion is a ***single asset staking ecosystem*** that provides yield back to its participants. The revenue is derived from diverse streams, internal and external to the ecosystem. This includes marketplace fees from Fusion, yield from external investments and trading, protocol fees from sister projects, and other sources.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

- High - Funds are ***directly*** at risk, or ***severe*** disruption of the protocol's core functionality
- Medium - Funds are ***indirectly*** at risk, or ***some*** disruption of the protocol's functionality
- Low - Funds are ***not*** at risk

Likelihood - the chance that a particular vulnerability gets discovered and exploited

- High - Highly likely to occur
- Medium - Might occur under specific conditions
- Low - Unlikely to occur

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 272db20d163dc9b16b691bd98cef9cd88e10e833**

Scope

The following smart contracts were in scope of the audit:

- `MicrogridNFT`
- `MicrogridNFTDeposit`
- `MarketplaceExchange`
- `MarketplaceInteract`
- `MicrogridBatteryManager`
- `MicrogridBatterySplitMyPosition`
- `BatteryInteractSplitMyPosition`
- `MicrogridBatteryWBNB`
- `MicrogridBatteryWETH`
- `BatteryInteractWBNB`
- `BatteryInteractWETH`
- `FusionRewardDistributor`
- `FusionAutoClaimUpkeep`
- `MigrateMasterNode`
- `ExchangeRateHelper`
- `OldFusionRewardDistributor`
- `ERC721A`

Out of scope:

- third-party libraries and dependencies
- other protocol contracts not listed above
- economic attacks

Test Approach

Using a combination of manual review and static analysis tools

Issues Found

The following number of issues were found, categorized by their severity:

- Critical & High: 24 issues
- Medium: 14 issues
- Low: 4 issues
- Informational: 5 issues

Recommend the EMP Fusion team review these findings in order of severity and pursue fixes, according to their specific remediations within this report.

Findings Summary

- ## Critical Risk Findings
  - [C-01] It's impossible for a user to claim his rewards, as `performUpkeep`, `claimFor`, `claimForMany` leaves rewards on contracts `BatteryInteractWETH` and `BatteryInteractWBNB`
  - [C-02] Wrong receiver's implementation in `claimFor` will result in loss of rewards
  - [C-03] User's rewards to be lost until the method `compoundFor` is called for the first time
  - [C-04] Malicious user can manipulate with adding and removing receivers, which will lead to incorrect calculation of rewards
  - [C-05] Calculation in `addPoints` reduces the user's balance
  - [C-06] Calculation for `finalAmount` will result in wrong decimals
  - [C-07] Wrong calculation for `finalAmount` may lead to loss of user funds
  - [C-08] Logic of `deposit` under conditions of `Sacrifice` is not implemented
  - [C-09] Wrong calculation for `amountWithBonus` may lead to loss of user funds
  - [C-10] Mint to an incorrectly specified address makes it impossible to buy a Battery NFT
  - [C-11] Accounting and updating `last_distPoints` allow malicious users to receive additional rewards
  - [C-12] Calculation for `ExchangeRate` will result in wrong decimals
  - [C-13] It's impossible for a user to claim his rewards, as `run` and `runFromUpkeep` will not pass `currentMinClaim` check
  - [C-14] Wrong calculation in `buyOrder` increases the `refundAmount`
  - [C-15] Missing a method for setting `MarketplaceContract`
- ## High Risk Findings

- Medium Risk Findings

- Low Risk Findings

- Informational

Detailed Findings

[C-01] It's impossible for a user to claim his rewards, as `performUpkeep`, `claimFor`, `claimForMany` leaves rewards on contracts `BatteryInteractWETH` and `BatteryInteractWBNB`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionAutoClaimUpkeep.sol#L44

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L251

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L275

Severity

Impact: High, because users will never receive rewards from the contract

Likelihood: High, as this will happen any time the method is used

Description

The `performUpkeep` method should send rewards for each user using `Battery` contract logic. However, this does not work, since in method `claimForMany` on contract `FusionRewardDistributor` the receivers are contracts `BatteryInteractWETH` and `BatteryInteractWBNB`. (The same situation occurs when calling methods `claimFor` or `claimForMany` directly). Since method `run` is not called further on contracts `BatteryInteractWETH` and `BatteryInteractWBNB`, all rewards remain on these contracts and will not distributed further.

This is wrong because it leads to loss of rewards.

Recommendations

Revise method `performUpkeep` to include logic related to further distribution of rewards to users. Restrict direct use of methods `claimFor`, `claimForMany`.

[C-02] Wrong receiver's implementation in `claimFor` will result in loss of rewards

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L251 https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L95

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L123

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L428

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryManager.sol#L64

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryManager.sol#L12

Severity

Impact: High, as this will result in 0 rewards for users when they try to use methods `claimFor` or `claimForMany`

Likelihood: High, as this will happen any time the methods are used

Description

The `claimFor` method sends user's rewards to the address of `_receiver` (the receivers are the `MicrogridBatteryWBNB` or `MicrogridBatteryWETH` contracts)

```
eshare.safeTransfer(_receiver, rewards);
```

However, for further sending to users, rewards must be sent to the corresponding contracts `BatteryInteractWBNB` and `BatteryInteractWETH` (which have the appropriate logic for this). The issue is that in the current edition, all rewards remain on the `MicrogridBatteryWBNB` or `MicrogridBatteryWETH` contracts and will not distributed further.

Recommendations

Add the mapping for pairs `MicrogridBattery` - `BatteryInteract` and AddressSet `_listOfReceiversInteractContracts` in contract `FusionRewardDistributor`

```
+ mapping(address => address) public receiversInteractContracts;
+ EnumerableSet.AddressSet private _listOfReceiversInteractContracts;
```

Change the code in the following way:

```
- function _setReceiver(uint256 _microgridNftId, uint256 _allocPoints,
address _receiver) internal {
+ function _setReceiver(uint256 _microgridNftId, uint256 _allocPoints,
address _receiver, address _receiverInteractContracts ) internal {
    require(verifiedReceivers[_receiver], "Caller is not verified
receiver");

    totalReceiversAllocPoints[_microgridNftId] =
totalReceiversAllocPoints[_microgridNftId].sub(receiversAllocPoints[_micro
gridNftId][_receiver]).add(
      _allocPoints
    );
    receiversAllocPoints[_microgridNftId][_receiver] = _allocPoints;
+   receiversInteractContracts[_receiver] = _receiverInteractContracts;
+   if
(!_listOfReceiversInteractContracts.contains(_receiverInteractContracts))
_listOfReceiversInteractContracts.add(_receiverInteractContracts);
    if (last_distPoints[_microgridNftId][_receiver] == 0) {
      last_distPoints[_microgridNftId][_receiver] = totalDistributePoints;
    }

    if (_allocPoints > 0) {
      if (!activeReceivers[_microgridNftId].contains(_receiver)) {
        activeReceivers[_microgridNftId].add(_receiver);
      }
      if (!activeTokenIds.contains(_microgridNftId)) {
        activeTokenIds.add(_microgridNftId);
      }
    } else {
      if (activeReceivers[_microgridNftId].contains(_receiver)) {
        activeReceivers[_microgridNftId].remove(_receiver);
      }
      if (activeTokenIds.contains(_microgridNftId)) {
        activeTokenIds.remove(_microgridNftId);
      }
    }
  }


- function setReceivers(uint256 _microgridNftId, uint256[] calldata
_allocPoints, address[] calldata _receivers) public {
+ function setReceivers(uint256 _microgridNftId, uint256[] calldata
_allocPoints, address[] calldata _receivers, address[] calldata
_receiverInteractContracts) public {
    require(msg.sender == batteryManagerContract, "Caller must be the
battery manager contract.");
    for (uint256 i = 0; i < _receivers.length; i++) {
-     _setReceiver(_microgridNftId, _allocPoints[i], _receivers[i]);
+     _setReceiver(_microgridNftId, _allocPoints[i], _receivers[i],
_receiverInteractContracts[i]);
    }
  }
  function claimFor(uint256 _id, address _receiver, bool _saveGas) public
whenNotPaused {
    if (receiversAllocPoints[_id][_receiver] > 0) {
      if (last_distPoints[_id][_receiver] == 0) {
        last_distPoints[_id][_receiver] = totalDistributePoints;
      }
      if (!_saveGas) {
        _tryClaimFarm();
      }

      uint256 rewards = _getDistributionRewards(_id, _receiver);

      if (rewards > 0) {
        esharePending = esharePending.sub(rewards);
        last_distPoints[_id][_receiver] = totalDistributePoints;

-       eshare.safeTransfer(_receiver, rewards);
+       eshare.safeTransfer(receiversInteractContracts[_receiver],
rewards);

        if (!_saveGas) {
          emit Claim(_id, rewards);
        }
      }
    }
  }
```

```
  function listClaimableReceivers(uint256 _startIndex, uint256 _endIndex)
external view returns (uint256[] memory, address[][] memory) {
    // Ensure _endIndex is within the bounds of the active token IDs array
    _endIndex = Math.min(_endIndex + 1, activeTokenIds.length());

    // Initialize temporary arrays to hold token IDs and receivers
    uint256[] memory tokenIds = new uint256[](_endIndex - _startIndex);
    address[][] memory temp = new address[][](_endIndex - _startIndex);

    uint256 counter = 0; // A counter to keep track of how many eligible
receivers we've found

    // Loop over the specified range of token IDs
    for (uint256 i = _startIndex; i < _endIndex; i++) {
      uint256 _tokenId = activeTokenIds.at(i);
      uint256 _numReceivers = activeReceivers[_tokenId].length();

      // Initialize an array to hold the eligible receivers for the
current token ID
      address[] memory eligibleReceivers = new address[](_numReceivers);
      uint256 eligibleCount = 0;

      // Loop over all receivers for the current token ID
      for (uint256 j = 0; j < _numReceivers; j++) {
        address _receiver = activeReceivers[_tokenId].at(j);

        // If the receiver can claim rewards, add them to the
eligibleReceivers array
-       if (getTotalRewards(_tokenId, _receiver) >=
IBattery(_receiver).minClaimAmount()) {
+       if (getTotalRewards(_tokenId, _receiver) >=
IBattery(receiversInteractContracts[_receiver]).minClaimAmount()) {
          eligibleReceivers[eligibleCount++] = _receiver;
        }
      }

        ........................
```

For contract `MicrogridBatteryManager` change the code in the following way:

```
- function activateBattery(uint256[] calldata _allocPoints, address[]
calldata _microgridBatteryNFTContracts) public {
+ function activateBattery(uint256[] calldata _allocPoints, address[]
calldata _microgridBatteryNFTContracts, address[] calldata
_interactBatteryContracts) public {
    // Find user's Microgrid token ID and battery token ID.
    uint256 usersMicrogridToken =
microgridContract.tokenByWallet(msg.sender);
    require(usersMicrogridToken > 0, "You must own a Microgrid NFT.");

    BatteryInfo storage batteryInfo =
batteryInfoByTokenId[usersMicrogridToken];

-  fusionDistributorContract.setReceivers(usersMicrogridToken,
_allocPoints, _microgridBatteryNFTContracts);
+  fusionDistributorContract.setReceivers(usersMicrogridToken,
_allocPoints, _microgridBatteryNFTContracts, _interactBatteryContracts);

    for (uint256 i = 0; i < _microgridBatteryNFTContracts.length; i++) {
      bool ownsBattery =
(IMicrogridBatteryNFT(_microgridBatteryNFTContracts[i])).checkBattery(user
sMicrogridToken);

        for (uint256 i = 0; i < _microgridBatteryNFTContracts.length; i++)
{
            bool ownsBattery =
(IMicrogridBatteryNFT(_microgridBatteryNFTContracts[i])).checkBattery(user
sMicrogridToken);
      require(ownsBattery == true, "Your Microgrid does not own this
battery.");

      if (batteryInfo.ownsBatteries[_microgridBatteryNFTContracts[i]] ==
false) {

batteryInfo.ownedBatteries.push(_microgridBatteryNFTContracts[i]);
          batteryInfo.ownsBatteries[_microgridBatteryNFTContracts[i]] ==
true;
      }
      batteryInfo.batteryPercents[_microgridBatteryNFTContracts[i]] =
_allocPoints[i];

      emit ActivateBattery(usersMicrogridToken, _allocPoints[i] > 0);
    }
  }
```

```diff
  interface IFusionRewardDistributor {
-   function setReceivers(uint256 _microgridNftId, uint256[] calldata
_allocPoints, address[] calldata _receivers) external;
+   function setReceivers(uint256 _microgridNftId, uint256[] calldata
_allocPoints, address[] calldata _receivers, address[] calldata
_receiverInteractContracts) external;
  }
```

[C-03] User's rewards to be lost until the method `compoundFor` is called for the first time

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L216

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L243

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L212

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: High, as this will happen with each user

Description

Each new user has `last_distPoints == 0`. When the method `compoundFor` (or `compoundForMany`, `compound`) is called for the first time, the `last_distPoints` is initialized (and in fact, only from this moment does the counting of rewards for a given user start). In according with this code:

```
  function compoundFor(uint256 _id, bool _claimBefore) public
whenNotPaused {
    if (last_distPoints[_id][address(this)] == 0) {
```

```
    last_distPoints[_id][address(this)] = totalDistributePoints;
  }
  if (_claimBefore) {
    _tryClaimFarm();
  }
  .........
```

This causes the user's rewards to be lost until the methods are called for the first time.

Recommendations

It is necessary to revise the logic so that when a user is added into protocol, method `_tryClaimFarm` is executed and `last_distPoints` in `FusionRewardDistributor` is initialized

```
last_distPoints[_id][address(this)] = totalDistributePoints;
```

[C-04] Malicious user can manipulate with adding and removing receivers, which will lead to incorrect calculation of rewards

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L95

Severity

Impact: High, as it will result in wrong reward calculations

Likelihood: High, as it requires no preconditions

Description

The method `_setReceiver` should initialize the `last_distPoints` when adding a new receiver as it's implemented with this code:

```
if (last_distPoints[_microgridNftId][_receiver] == 0) {
  last_distPoints[_microgridNftId][_receiver] = totalDistributePoints;
}
```

But the user can remove their previously initialized receiver and then add it again (in which case the initialization will not occur).

Malicious user can initialize 2 receivers (MicrogridBatteryWBNB and MicrogridBatteryWETH). Then remove one receiver. When distributing rewards, 100% will be sent to the second receiver. After which the user will remove the second one, connect the first one and again receive all the rewards on the first receiver.

This way the user will be able to receive extra rewards.

Recommendations

Need to initialize the receiver every time it is added. Change the code in the following way:

```
- if (last_distPoints[_microgridNftId][_receiver] == 0) {
    last_distPoints[_microgridNftId][_receiver] = totalDistributePoints;
- }
```

[C-05] Calculation in addPoints reduces the user's balance

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceExchange.sol#L100

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: High, as this will happen any time the user buys points

Description

The method addPoints calculates incorrectly. When adding a purchaseAmount it reduces the user's balance.

Recommendations

Change the code in the following way:

```
  function addPoints(uint256 microgridId, uint256 purchaseAmount) external {
    require(allowedCreditors[msg.sender] == true, "Only allowed creditors can add points.");
-   userBalance[microgridId] -= purchaseAmount;
+   userBalance[microgridId] += purchaseAmount;
  }
```

[C-06] Calculation for finalAmount will result in wrong decimals

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L932

Severity

Impact: High, as it will result in wrong points calculations

Likelihood: High, as this will happen any time the user buys points

Description

The method buyPoints is implemented so that

- during calculations finalAmount is obtained with a decimals 1e18 times large than necessary (in cases token == address(empToken) or allowedCurrencies[token].exRateHelper == true)
- during calculations finalAmount is obtained with a decimals 1e18 times less than necessary (in other cases)

Recommendations

Change the code in the following way:

```
  // If token is being spent to purchase Marketplace Points.
  if (token != address(0)) {
   IERC20(token).transferFrom(msg.sender, address(marketplaceContract), amount);
   if (token == address(empToken)) {
-    uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token) * 1e18;
+    uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token);
    uint256 purchaseAmount = ((((currentPrice * amount) / 1e18) * allowedCurrencies[token].pointsPerDollar) * 10000) * (calcRateEMP() / 1e18) / 10000;
    uint256 bonusPercent = getBonusPercent(purchaseAmount);
    uint256 finalAmount = ((((currentPrice * amount) / 1e18) * allowedCurrencies[token].pointsPerDollar) * 10000) * (calcRateEMP() / 1e18) / 10000 * (bonusPercent * 10000) / 10000;
    marketplaceContract.addPoints(usersMicrogridId, finalAmount);
    emit PointsBought(usersMicrogridId, finalAmount);
   } else if (allowedCurrencies[token].exRateHelper == true) {
-    uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token) * 1e18;
+    uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token);
    uint256 purchaseAmount = ((currentPrice * amount) / 1e18) * allowedCurrencies[token].pointsPerDollar;
    uint256 bonusPercent = getBonusPercent(purchaseAmount);
```

```
        uint256 finalAmount = ((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar * (bonusPercent * 10000) / 10000;
        marketplaceContract.addPoints(usersMicrogridId, finalAmount);
        emit PointsBought(usersMicrogridId, finalAmount);
      } else {
        int256 currentPrice =
IAggregator(allowedCurrencies[token].priceAggregator).latestAnswer();
-       uint256 purchaseAmount = (((amount * (uint256(currentPrice) / 10
** IAggregator(allowedCurrencies[token].priceAggregator).decimals())) /
1e18) * allowedCurrencies[token].pointsPerDollar);
+       uint256 purchaseAmount = ((amount * uint256(currentPrice) / 10 **
IAggregator(allowedCurrencies[token].priceAggregator).decimals()) *
allowedCurrencies[token].pointsPerDollar);
        uint256 bonusPercent = getBonusPercent(purchaseAmount);
        uint256 finalAmount = (((amount * (uint256(currentPrice) / 10 **
IAggregator(allowedCurrencies[token].priceAggregator).decimals())) / 1e18)
* allowedCurrencies[token].pointsPerDollar) * (bonusPercent *   10000) /
10000;
        marketplaceContract.addPoints(usersMicrogridId, finalAmount);
        emit PointsBought(usersMicrogridId, finalAmount);
      }

    // Else BNB is being spent.
    } else {
      payable(address(marketplaceContract)).transfer(msg.value);
      int256 currentPrice =
IAggregator(aggregatorContract).latestAnswer();
-     uint256 purchaseAmount = (((msg.value * (uint256(currentPrice) / 10
** IAggregator(aggregatorContract).decimals())) / 1e18) *
pointsPerDollar);
+     uint256 purchaseAmount = ((msg.value * uint256(currentPrice) / 10 **
IAggregator(aggregatorContract).decimals()) * pointsPerDollar);
      uint256 bonusPercent = getBonusPercent(purchaseAmount);
      uint256 finalAmount = (((msg.value * (uint256(currentPrice) / 10 **
IAggregator(aggregatorContract).decimals())) / 1e18) * pointsPerDollar) *
(bonusPercent * 10000) / 10000;
      marketplaceContract.addPoints(usersMicrogridId, finalAmount);
      emit PointsBought(usersMicrogridId, finalAmount);
    }
```

[C-07] Wrong calculation for `finalAmount` may lead to loss of user funds

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L932

Severity

Impact: High, as it will result in wrong points calculations and loss of user funds

Likelihood: High, as this will happen any time the user buys points

Description

The method `buyPoints` uses `bonusPercent`. But calculations are implemented incorrectly, which can lead to loss of user funds (for example, `with bonusPercent == 0`).

Recommendations

Change the code in the following way:

```
    // If token is being spent to purchase Marketplace Points.
   if (token != address(0)) {
    IERC20(token).transferFrom(msg.sender, address(marketplaceContract),
amount);
    if (token == address(empToken)) {
      uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token) *
1e18;
      uint256 purchaseAmount = ((((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar) * 10000) * (calcRateEMP() /
1e18) / 10000;
      uint256 bonusPercent = getBonusPercent(purchaseAmount);
-     uint256 finalAmount = ((((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar) * 10000) * (calcRateEMP() /
1e18) / 10000 * (bonusPercent * 10000) / 10000;
+     uint256 finalAmount = ((((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar) * 10000) * (calcRateEMP() /
1e18) / 10000 * (bonusPercent + 10000) / 10000;
      marketplaceContract.addPoints(usersMicrogridId, finalAmount);
      emit PointsBought(usersMicrogridId, finalAmount);
    } else if (allowedCurrencies[token].exRateHelper == true) {
      uint256 currentPrice = microgridNFTDeposit.getExchangeRate(token) *
1e18;
      uint256 purchaseAmount = ((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar;
      uint256 bonusPercent = getBonusPercent(purchaseAmount);
-     uint256 finalAmount = ((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar * (bonusPercent * 10000) / 10000;
+     uint256 finalAmount = ((currentPrice * amount) / 1e18) *
allowedCurrencies[token].pointsPerDollar * (bonusPercent + 10000) / 10000;
      marketplaceContract.addPoints(usersMicrogridId, finalAmount);
      emit PointsBought(usersMicrogridId, finalAmount);
    } else {
      int256 currentPrice =
IAggregator(allowedCurrencies[token].priceAggregator).latestAnswer();
      uint256 purchaseAmount = (((amount * (uint256(currentPrice) / 10 **
IAggregator(allowedCurrencies[token].priceAggregator).decimals())) / 1e18)
* allowedCurrencies[token].pointsPerDollar);
      uint256 bonusPercent = getBonusPercent(purchaseAmount);
-     uint256 finalAmount = (((amount * (uint256(currentPrice) / 10 **
IAggregator(allowedCurrencies[token].priceAggregator).decimals())) / 1e18)
* allowedCurrencies[token].pointsPerDollar) * (bonusPercent *   10000) /
10000;
+     uint256 finalAmount = ((amount * uint256(currentPrice) / 10 **
IAggregator(allowedCurrencies[token].priceAggregator).decimals()) *
allowedCurrencies[token].pointsPerDollar) * (bonusPercent + 10000) /
10000;
      marketplaceContract.addPoints(usersMicrogridId, finalAmount);
      emit PointsBought(usersMicrogridId, finalAmount);
    }

    // Else BNB is being spent.
    } else {
      payable(address(marketplaceContract)).transfer(msg.value);
      int256 currentPrice =
IAggregator(aggregatorContract).latestAnswer();
      uint256 purchaseAmount = (((msg.value * (uint256(currentPrice) / 10
** IAggregator(aggregatorContract).decimals())) / 1e18) *
pointsPerDollar);
      uint256 bonusPercent = getBonusPercent(purchaseAmount);
-     uint256 finalAmount = (((msg.value * (uint256(currentPrice) / 10 **
IAggregator(aggregatorContract).decimals())) / 1e18) * pointsPerDollar) *
(bonusPercent * 10000) / 10000;
+     uint256 finalAmount = ((msg.value * uint256(currentPrice) / 10 **
IAggregator(aggregatorContract).decimals()) * pointsPerDollar) *
(bonusPercent + 10000) / 10000;
      marketplaceContract.addPoints(usersMicrogridId, finalAmount);
      emit PointsBought(usersMicrogridId, finalAmount);
    }
```

[C-08] Logic of `deposit` under conditions of `Sacrifice` is not implemented

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1255

Severity

Impact: High, as this will lead to restriction on the functionality of protocol

Likelihood: High, as this will happen any time the user try to deposit with sacrifice

Description

The logic of the protocol assumes that in method `deposit` in cases `DepositType(depositType) == DepositType.BY_SACRIFICE` and `DepositType(depositType) == DepositType.BY_SACRIFICE_USD` a deposit will be carried out without the actual transfer of tokens (or

BNB) from users to the contract `MicrogridNFT`. The current version of the code does not implement this, and the transaction will revert.

Recommendations

Change the code in the following way:

```
    if ((DepositType(depositType) == DepositType.BY_EMP_ETH_LP)) {
      empEthLpToken.transferFrom(tx.origin, address(microgridNFTContract), 
amount);

    } else if ((DepositType(depositType) == DepositType.DEFAULT)) {
      empToken.transferFrom(msg.sender, address(microgridNFTContract), 
amount);

    } else if ((DepositType(depositType) == DepositType.BY_UPEMP)) {
      upEmp.transferFrom(tx.origin, address(microgridNFTContract), 
amount);

    } else if ((DepositType(depositType) == DepositType.CURRENCY ||
      DepositType(depositType) == DepositType.CURRENCY_BY_TOKEN_ID) &&
      currency != address(0)) {
      (IERC20(currency)).transferFrom(msg.sender, 
address(microgridNFTContract), amount);

+   } else if ((DepositType(depositType) == DepositType.BY_SACRIFICE ||
+     DepositType(depositType) == DepositType.BY_SACRIFICE_USD)) {

    } else if (msg.value == 0){
      empToken.transferFrom(msg.sender, address(microgridNFTContract), 
amount);

    } else {
      payable(address(microgridNFTContract)).transfer(msg.value);
    }
    ...........
```

[C-09] Wrong calculation for `amountWithBonus` may lead to loss of user funds

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1255

Severity

Impact: High, as it will result in wrong deposit calculations and loss of user funds

Likelihood: High, as this will happen any time the user deposit

Description

The method `deposit` in case of `DepositType(depositType) == DepositType.BY_SACRIFICE_USD` uses `currentPrice` of `empToken`. But calculations are implemented incorrectly.

Also, in cases `DepositType(depositType) == DepositType.CURRENCY` or `DepositType(depositType) == DepositType.CURRENCY_BY_TOKEN_ID`, there may be a situation that `getExchangeRate(address(empToken)) > getExchangeRate(address(currency))`. As a result of calculations, `currentPrice == 0`.

This can lead to loss of user funds.

Recommendations

Change the code in the following way:

```
    if (DepositType(depositType) == DepositType.DEFAULT ||
      DepositType(depositType) == DepositType.BY_TOKEN_ID ||
      DepositType(depositType) == DepositType.FOR_NEW_USER) {
      amountWithBonus = (amountWithBonus * 10000) * (calcRateEMP() / 1e18)
/ 10000;

    } else if (DepositType(depositType) == DepositType.BY_SACRIFICE) {
      amountWithBonus = (amountWithBonus *
allowedSacrifices[msg.sender].sacMultiplier) / 10000;

    } else if (DepositType(depositType) == DepositType.BY_SACRIFICE_USD) {
      uint256 currentPrice = (getExchangeRate(address(empToken)));
-     amountWithBonus = (((amountWithBonus * currentPrice) / 1e18) *
allowedSacrifices[msg.sender].sacMultiplier) / 10000;
+     amountWithBonus = (((amountWithBonus / currentPrice) / 1e18) *
allowedSacrifices[msg.sender].sacMultiplier) / 10000;

    } else if ((DepositType(depositType) == DepositType.BY_UPEMP)) {
      amountWithBonus = (upEmp.calculatePrice() / 1e18) * amountWithBonus;

    } else if ((DepositType(depositType) == DepositType.BY_EMP_ETH_LP)) {
      amountWithBonus = ((amountWithBonus * 2 * 1e18 *
empToken.balanceOf(address(empEthLpToken)) / empEthLpToken.totalSupply() /
1e18) * 10000) * (calcRateEMP() / 1e18) / 10000;

    } else if (DepositType(depositType) == DepositType.CURRENCY ||
      DepositType(depositType) == DepositType.CURRENCY_BY_TOKEN_ID) {
      int256 currentPrice = 0;
      uint256 _sharesPerEMP = sharesPerEMP;

      if (currency == address(0)) {
-      currentPrice = (int256(getExchangeRate(currency))) /
(int256(getExchangeRate(address(empToken))));
+      currentPrice = (int256(getExchangeRate(currency) * 1e18)) /
(int256(getExchangeRate(address(empToken))));
      } else {
-      currentPrice = (int256(getExchangeRate(currency))) /
(int256(getExchangeRate(address(empToken))));
+      currentPrice = (int256(getExchangeRate(currency) * 1e18)) /
(int256(getExchangeRate(address(empToken))));
        _sharesPerEMP = allowedCurrencies[currency].sharesPerEMP;
      }

      amountWithBonus= (amountWithBonus * ((uint256(currentPrice)) *
_sharesPerEMP) / 1e18) / 1e18;
    }
```

[C-10] Mint to an incorrectly specified address makes it impossible to buy a Battery NFT

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1291

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1292

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L221

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: High, as this will happen any time the user call method

Description

The method `buyBattery` performs `mint` of `Battery NFT` via external call `batteryContract.mint` and the `address(batteryContract)` is passed as a parameter.

On the `Battery` contract `mint` is implemented by the following code:

```
  function mint(address user, uint256 amount) external {
    uint256 usersMicrogridToken = microgridContract.tokenByWallet(user);
    require(interactContract == msg.sender, "Only the interact contract
can mint.");
    require(ownsBattery[usersMicrogridToken] == false, "You already own
this battery.");
    ownsBattery[usersMicrogridToken] = true;
    _mint(address(this), amount);
  }
```

Therefore, msg.sender must be passed as a parameter. The current implementation results in the loss of user funds, since the NFT is not issued to his address.

Functions with this issue:

- WBNBBatteryInteract.buyBattery
- WETHBatteryInteract.buyBattery
- BatteryInteractSplitMyPosition.buyBattery

Recommendations

Change the code in the following way:

```
    // Purchase battery w/Marketplace Points.
      marketplaceContract.spendPoints(usersMicrogridToken, batteryCost);

    // Mint NFT.
-     batteryContract.mint(address(batteryContract), 1);
+     batteryContract.mint(msg.sender, 1);

     ..........
```

[C-11] Accounting and updating last_distPoints allow malicious users to receive additional rewards

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: High, since multiple attack vectors are possible

Description

The DistributePoints value for each user's receiver is saved to mapping last_distPoints. These values are updated on the contract FusionRewardDistributor

```
      last_distPoints[_id][_receiver] = totalDistributePoints;
```

This happens at:

- adding a new `Battery receiver` (`_setReceiver`)
- calling the method `compoundFor`

```
  function compoundFor(uint256 _id, bool _claimBefore) public
whenNotPaused {
    if (last_distPoints[_id][address(this)] == 0) {
      last_distPoints[_id][address(this)] = totalDistributePoints;
    }
    if (_claimBefore) {
      _tryClaimFarm();
    }

    if (totalReceiversAllocPoints[_id] == 0) {
      uint256 distributed = _getDistributionRewards(_id, address(this));

      if (distributed > 0) {
        last_distPoints[_id][address(this)] = totalDistributePoints;
        esharePending = esharePending.sub(distributed);
        eshareCompounded = eshareCompounded.add(distributed);

        nft.compoundForTokenId(distributed, _id);

        _tryClaimFarm();

        if (last_distPoints[_id][address(this)] != totalDistributePoints)
last_distPoints[_id][address(this)] = totalDistributePoints;

        emit Compound(_id, distributed);
      }
    }
  }
```

Moreover, the update occurs only in the case of `totalReceiversAllocPoints == 0` (if user does not have a `Battery receiver`)

- calling the method `claimFor`

```
  function claimFor(uint256 _id, address _receiver, bool _saveGas) public
whenNotPaused {
    if (receiversAllocPoints[_id][_receiver] > 0) {
      if (last_distPoints[_id][_receiver] == 0) {
        last_distPoints[_id][_receiver] = totalDistributePoints;
      }
      if (!_saveGas) {
        _tryClaimFarm();
      }

      uint256 rewards = _getDistributionRewards(_id, _receiver);

      if (rewards > 0) {
        esharePending = esharePending.sub(rewards);
        last_distPoints[_id][_receiver] = totalDistributePoints;

        eshare.safeTransfer(_receiver, rewards);

        if (!_saveGas) {
        emit Claim(_id, rewards);
        }
      }
    }
  }
```

This creates the following attack vectors:

1. Via Deposit

```
The malicious user has

  MicrogridNFT id = 1

  individualShares = 100

  MicrogridBatteryWBNB - receiver

  receiversAllocPoints for user’s MicrogridBatteryWBNB = 1000

  totalReceiversAllocPoints for user’s = 1000

  last_distPoints[1][address(MicrogridBatteryWBNB)] = 50000
```

At the same time, on the contract `totalDistributePoints = 50000`.

The malicious user adds another `200 shares` via `deposit` on the `MicrogridNFTDeposit` contract (`addSharesByDeposit` on `MicrogridNFT`)

```
  function addSharesByDeposit(uint256 amount, uint256 tokenId) external
onlyDepositContract {
    require(_exists(tokenId), "Token does not exist");

    individualShares[tokenId] += amount;
    totalShares += amount;
    _updateRewardsAndPoints(tokenId);
  }
```

After adding, the method `updateRewardsAndPonts` is called on `FusionRewardDistributor`

```
  function updateRewardsAndPoints(uint256 _id) external {
    require(_msgSender() == address(nft), "Caller is not MicroGridNFT");
    _tryClaimFarm();
    compoundFor(_id, false);
  }
```

This is where `rewards` are received from `Farm` (`_tryClaimFarm`) and `compoundFor` is called.

New value on the contract is `totalDistributePoints = 80000`.

However, the `compound` for malicious user does not occur (since `totalReceiversAllocPoints != 0`) and there is no update to the `last_distPoints` value either.

```
New values

  individualShares = 300

  MicrogridBatteryWBNB – receiver

  receiversAllocPoints for user’s MicrogridBatteryWBNB = 1000

  totalReceiversAllocPoints for user’s = 1000

  last_distPoints[1][address(MicrogridBatteryWBNB)] = 50000
```

Next, malicious user calls `claimFor`, where the `rewards` for `receiver` are calculated using `_getDistributionRewards` as follows

```
  function _getDistributionRewards(uint256 _id, address _receiver)
internal view returns (uint256) {
    uint256 _points = last_distPoints[_id][_receiver];
    if (_points == 0) return 0;

    uint256 _distributionRewards =
nft.individualShares(_id).mul(totalDistributePoints.sub(_points)).div(MULT
IPLIER);
    uint256 _receiverAllocPoints = receiversAllocPoints[_id][_receiver];
    uint256 _totalReceiversAllocPoints = totalReceiversAllocPoints[_id];

    if (_totalReceiversAllocPoints == 0 && _receiver == address(this))
return _distributionRewards;

    return
      verifiedReceivers[_receiver] && _receiverAllocPoints > 0 &&
_totalReceiversAllocPoints > 0
        ?
_distributionRewards.mul(_receiverAllocPoints).div(_totalReceiversAllocPoi
nts)
        : 0;
  }
```

Here you can see that malicious user will receive `rewards` based on the new `individualShares` value - 3 times more than they should have.

In one option, the `attacker` can monitor the mempool in order to make a frontrun transaction that outputs rewards from `Farm` to `FusionRewardDistributor` (while sharply increasing the value of its `individualShares`).

An additional problem for the protocol would be that if, due to this situation, it turns out that `eSharePending < distributed`

```
      esharePending = esharePending.sub(distributed);
```

2. Via Split

```
Let's assume there are 2 attackers with same balances

  MicrogridNFT id

  individualShares = 100

  MicrogridBatteryWBNB – receiver

  receiversAllocPoints for user’s MicrogridBatteryWBNB = 1000

  totalReceiversAllocPoints for user’s = 1000

  last_distPoints[id][address(MicrogridBatteryWBNB)] = 50000

  MicrogridBatterySplitMyPosition
```

At the same time, on the contract `totalDistributePoints = 50000`.

Attacker1 puts 100 of his `individualShares` up for sale using the split method on `BatteryInteractSplitMyPosition`

```
  function split(uint256 sharesAmount, uint256 pricePerShare) public
nonReentrant {
    // Find user's microgrid token ID.
    uint256 usersMicrogridToken =
microgridContract.tokenByWallet(msg.sender);

    // Requires
    require(usersMicrogridToken > 0, "You must own a Microgrid NFT.");
    require(sharesAmount <=
microgridContract.individualShares(usersMicrogridToken), "You cannot split
more shares than you own.");

    // Remove the sharesAmount from user's individualShares temporarily.
    uint256 currentShares =
microgridContract.individualShares(usersMicrogridToken);
    uint256 newShares = currentShares - sharesAmount;
    microgridContract.setShares(newShares, usersMicrogridToken);

    // Add the sharesAmount temporarily to the Treasury wallet.
    uint256 treasuryMicrogridToken =
microgridContract.tokenByWallet(treasury);
    uint256 treasuryCurrentShares =
microgridContract.individualShares(treasuryMicrogridToken);
    uint256 treasuryNewShares = treasuryCurrentShares + sharesAmount;
    microgridContract.setShares(treasuryNewShares,
treasuryMicrogridToken);

    // Add the shares and pricePerShare to the salesList mapping.
    ..........................
    }
```

Here new `individualShares` values (0 for Attacker1, 100 for Treasury) are written via the method `setShares` on `MicrogridNFT`

```
  function setShares(uint256 amount, uint256 tokenId) external
nonReentrant {
    require(_exists(tokenId), "Token does not exist");
    require(
      allowedBatteries[msg.sender],
      "Only batteries can set a tokenId's shares."
    );

    uint256 previousShares = individualShares[tokenId];

    individualShares[tokenId] = amount;
    totalShares -= previousShares;
    totalShares += amount;
    _updateRewardsAndPoints(tokenId);
  }
```

After adding, the `updateRewardsAndPonts` method is called.

New value on the contract `totalDistributePoints = 80000`.

However, the `compound` for the user does not occur (since `totalReceiversAllocPoints != 0`) and there is no update to the `last_distPoints` value either.

After this, Attacker2, through the `buyOrder` method on `BatteryInteractSplitMyPosition`, acquires 100 `individualShares` (now he has 200).

```
  function buyOrder(uint256 orderId, uint256 sharesAmount) public payable
nonReentrant {
    // Find microgrid token ID and current Shares.
    uint256 usersMicrogridToken =
microgridContract.tokenByWallet(msg.sender);
    uint256 treasuryMicrogridToken =
microgridContract.tokenByWallet(treasury);
    uint256 buyerCurrentShares =
microgridContract.individualShares(usersMicrogridToken);
    uint256 treasuryCurrentShares =
microgridContract.individualShares(treasuryMicrogridToken);

    ............................

    // Transfer shares.
    uint256 buyerNewShares = buyerCurrentShares + sharesAmount;
    microgridContract.setShares(buyerNewShares, usersMicrogridToken);

    uint256 treasuryNewShares = treasuryCurrentShares - sharesAmount;
    microgridContract.setShares(treasuryNewShares,
treasuryMicrogridToken);

    // Emit event.
    emit SharesBought(orderId, sharesAmount, feeAmount + sellerAmount);
  }
```

Next, Attacker2 calls `claimFor`, receiving extra rewards.

The next step is to do this procedure in reverse order.

As a result, Attacker1 has

```
individualShares = 200

last_distPoints[id][address(MicrogridBatteryWBNB)] = 50000 (as in the
beginning)
```

After which Attacker1 also receives extra rewards.

We would also like to point out that with each `split` the `compound` is executed for Treasury (`totalReceiversAllocPoints == 0`), which leads to loss of user rewards.

Recommendations

We recommend

- changing the logic for accounting and updating `last_distPoints`
- add check that `eSharePending >= distributed`
- reviewing logic with compound for Treasury

[C-12] Calculation for `ExchangeRate` will result in wrong decimals

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1449

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1454

Severity

Impact: High, as it will result in wrong rate calculations

Likelihood: High, as this will happen any time the user claim rewards

Description

The method `_getExchangeRate` is implemented so that during calculations `Rate` is obtained with a decimals 1e18 times less than necessary.

Functions with this issue:

- `WBNBBatteryInteract._getExchangeRate`
- `WETHBatteryInteract._getExchangeRate`

Recommendations

Change the code in the following way:

```
  function _getExchangeRate(address token, uint256 usdValue) internal view
returns (uint256) {
    uint256 exchangeRate = microgridNFTDeposit.getExchangeRate(token);
    require(exchangeRate > 0, "not_listed");

    uint256 decimals = uint256(IERC20(token).decimals());

-   return (usdValue * 10**decimals / exchangeRate);
+   return (usdValue * 1e18 * 10**decimals / exchangeRate);
  }
```

[C-13] It's impossible for a user to claim his rewards, as `run` and `runFromUpkeep` will not pass `currentMinClaim` check

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1311

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1346

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1311

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1348

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L896

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L896

Severity

Impact: High, because users will not receive rewards from the contract

Likelihood: High, as this will happen any time the user claim rewards

Description

The `run` method should be used by a user to receive his rewards. This won't ever work due to the following issues in the code implementation:

At the beginning of the method, a check is made that the value of rewards is not less than the amount of tokens (`currentMinClaim`) obtained from recalculating `minClaimAmount` denominated in USD. However, the external call to `claimFor` on `FusionRewardDistributor`, which is responsible for receiving rewards from the `farm` and further distribution, occurs after this check.

- Also, `claimFor` is implemented in such a way that if the parameter `_saveGas == true` (as specified in method `run` implementation), the call to `_tryClaimFarm` is not carried out.

```
  function claimFor(uint256 _id, address _receiver, bool _saveGas)
public whenNotPaused {
    if (receiversAllocPoints[_id][_receiver] > 0) {
      if (last_distPoints[_id][_receiver] == 0) {
        last_distPoints[_id][_receiver] = totalDistributePoints;
      }
```

```
        if (!_saveGas) {
          _tryClaimFarm();
        }

        uint256 rewards = _getDistributionRewards(_id, _receiver);

        if (rewards > 0) {
          esharePending = esharePending.sub(rewards);
          last_distPoints[_id][_receiver] = totalDistributePoints;

          eshare.safeTransfer(_receiver, rewards);

          if (!_saveGas) {
            emit Claim(_id, rewards);
          }
        }
      }
    }
```

So checking of `currentMinClaim` will result in revert.

- At external call of method `getTotalRewards` on `FusionRewardDistributor`, the parameter `_receiver` must be `batteryContract` (not `address(this)`).
- The method `claimFor` on `FusionRewardDistributor` has 3 parameters (not 2, as specified in the code) - parameter `_receiver` is missing.

Functions with these issues:

- `WBNBBatteryInteract.run`
- `WBNBBatteryInteract.runFromUpkeep`
- `WETHBatteryInteract.run`
- `WETHBatteryInteract.runFromUpkeep`

Recommendations

Add functionality to ensure that rewards are transfered from the `farm` and distributed before checking.

Add the method `batteryClaimFarm` in contract `FusionRewardDistributor`

```
+   function batteryClaimFarm() external {
+     require(_listOfReceiversInteractContracts.contains(msg.sender),
"Caller is not ReceiversInteractContract");
+     _tryClaimFarm();
+     _distributeRewards();
+   }
```

Change the code in the following way:

```
    function run() public nonReentrant {
      // Find user's Microgrid token ID and current minimum claim amount.
      uint256 usersMicrogridToken =
microgridContract.tokenByWallet(msg.sender);
      uint256 currentMinClaim = _getExchangeRate(address(eshareToken),
minClaimAmount);

      // Requires and WBNB variable.
      require(usersMicrogridToken > 0, "You must own a Microgrid NFT.");
      require(batteryContract.checkBattery(usersMicrogridToken) == true,
"Your Microgrid does not own this battery.");
+     fusionDistributorContract.batteryClaimFarm();
-
require(fusionDistributorContract.getTotalRewards(usersMicrogridToken,
address(this)) >= currentMinClaim, "You don't have enough claimable
rewards yet.");
+
require(fusionDistributorContract.getTotalRewards(usersMicrogridToken,
address(batteryContract)) >= currentMinClaim, "You don't have enough
claimable rewards yet.");
      require(lastClaimTime[usersMicrogridToken] + claimTimeLimit <=
block.timestamp, "You have to wait for the claim time limit to pass.");
      IWBNB WBNB = IWBNB(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c);

      // Establish initial balances.
      uint256 beforeBalanceEshare = eshareToken.balanceOf(address(this));

      // Claim from fusion reward distributor.
-     fusionDistributorContract.claimFor(usersMicrogridToken, true);
+     fusionDistributorContract.claimFor(usersMicrogridToken,
address(batteryContract), true);

      // Get received Eshare amount and Sell 100% of it for WBNB.
      uint256 eshareSellAmount = eshareToken.balanceOf(address(this)) -
beforeBalanceEshare;
      .....................
```

```
    function runFromUpkeep(address holder) public nonReentrant {
      // Find user's Microgrid token ID and current minimum claim amount.
      uint256 usersMicrogridToken =
microgridContract.tokenByWallet(holder);
      uint256 currentMinClaim = _getExchangeRate(address(eshareToken),
minClaimAmount);

      // Requires and WBNB variable.
      require(msg.sender == upkeepContract, "Only the Upkeep contract can
use this function.");
      require(usersMicrogridToken > 0, "You must own a Microgrid NFT.");
      require(batteryContract.checkBattery(usersMicrogridToken) == true,
"Your Microgrid does not own this battery.");
+     fusionDistributorContract.batteryClaimFarm();
```

```
-
require(fusionDistributorContract.getTotalRewards(usersMicrogridToken,
address(this)) >= currentMinClaim, "You don't have enough claimable
rewards yet.");
+
require(fusionDistributorContract.getTotalRewards(usersMicrogridToken,
address(batteryContract)) >= currentMinClaim, "You don't have enough
claimable rewards yet.");
        require(lastClaimTime[usersMicrogridToken] + claimTimeLimit <=
block.timestamp, "You have to wait for the claim time limit to pass.");
        IWBNB WBNB = IWBNB(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c);

        // Establish initial balances.
        uint256 beforeBalanceEshare = eshareToken.balanceOf(address(this));

        // Claim from fusion reward distributor.
-       fusionDistributorContract.claimFor(usersMicrogridToken, true);
+       fusionDistributorContract.claimFor(usersMicrogridToken,
address(batteryContract), true);

        // Get received Eshare amount and Sell 100% of it for WBNB.
        uint256 eshareSellAmount = eshareToken.balanceOf(address(this)) -
beforeBalanceEshare;
        .....................
```

```
    interface IFusionRewardDistributor {
-       function claimFor(uint256 tokenId, bool claimBefore) external;
+       function claimFor(uint256 tokenId, address receiver, bool
claimBefore) external;
        function getTotalRewards(uint256 _tokenId, address _receiver)
external view returns (uint256);
        function setReceiver(uint256 _id, uint256 _allocPoints) external;
+       function batteryClaimFarm() external;
    }
```

[C-14] Wrong calculation in buyOrder increases the refundAmount

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L355

Severity

Impact: High, as this will lead to a monetary loss for protocol

Likelihood: High, as this will happen any time the user buys order

Description

The value of `refundAmount` in the method `buyOrder` calculates incorrectly.

Recommendations

Change the code in the following way:

```
      ................
      // Refund overpay amount, if buyer overpaid.
      if (msg.value > sellerAmount + feeAmount) {
-       uint256 refundAmount = msg.value - sellerAmount + feeAmount;
+       uint256 refundAmount = msg.value - sellerAmount - feeAmount;
        payable(msg.sender).transfer(refundAmount);
      }
      ................
```

[C-15] Missing a method for setting `MarketplaceContract`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol

Severity

Impact: High, as this will restrict the functionality of contract

Likelihood: High, as the contract cannot be used

Description

The `MicrogridBatterySplitMyPositionInteract` is missing a method for setting `MarketplaceContract`. This leads to the impossibility of using the method `buyBattery`, and as a result, disabling contract functionality.

Recommendations

Add a setter for `MarketplaceContract`.

[H-01] Missing method `receive` in contract

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceExchange.sol

Severity

Impact: Medium, as no value will be lost but the contract functionality will be limited

Likelihood: High, as the function will just revert every time

Description

The contract `MarketplaceExchange` is missing a method `receive`. This makes it impossible to receive BNB from the contract `MarketplaceInteract`, which violates the functionality of the method `buyPoints`.

Recommendations

Add method `receive` to the contract `MarketplaceExchange`.

[H-02] Missing a method for setting BNB `pointsPerDollar`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol

Severity

Impact: High, as this will lead to a monetary loss for users and restrict the functionality of contract

Likelihood: Medium, it affects user assets only with paying with BNB

Description

The `MarketplaceInteract` is missing a method for setting BNB `pointsPerDollar`. This can lead to the loss of user funds when calling the method `buyPoints` and paying with BNB.

Recommendations

Add a setter for BNB `pointsPerDollar`.

[H-03] Unused `payable` state mutability

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1291

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1292

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L221

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: Medium, as it requires a error on user's side

Description

The method has a `payable` state mutability, but nowhere in the code implementation is `msg.value` used. (Also, the logic of the method does not imply the use of payment in BNB). This may cause the user's BNB to get stuck in contract.

Functions with this issue:

- `WBNBBatteryInteract.buyBattery`
- `WETHBatteryInteract.buyBattery`
- `BatteryInteractSplitMyPosition.buyBattery`

Recommendations

Remove `payable` state mutability.

[H-04] Exchange Rate can be manipulated

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1202

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L906

Severity

Impact: High, as this will lead to a monetary loss for protocol

Likelihood: Medium, as it happens only in case of using the method as the main price feed

Description

A malicious user can manipulate the protocol to get more shares from the `MicrogridNFTDeposit` than they should. The method `deposit` for calculations uses `getExchangeRate` - in which a possible option is to get the price through `getRateFromDex`. The calculation uses values of `reserve0` and `reserve1` in LP pair (`IPancakePair`) that can be manipulated by flashLoan.

A similar method `getRateFromDex` is also implemented in the contract `ExchangeRateHelper`.

Recommendations

Add validation for price obtained from `getRateFromDex` using external oracles.

[H-05] `getExchangeRate` not take into account the possibility of `tokens` in `Pair` with different `decimals`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1202

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L906

Severity

Impact: High, as this will lead to a monetary loss for protocol

Likelihood: Medium, as it happens only in case of using the method as the main price feed

Description

In method `getExchangeRate` one of the possible options is to get the price through `getRateFromDex`. Implementation of `getExchangeRate` in a protocol assumes that it returns values with `decimals == 18`.

`getRateFromDex` to calculate `tokenPrice` uses values of `reserve0` and `reserve1` in LP pair (`IPancakePair`). However, tokens in a pair may have different `decimals` values (and therefore - `reserve0`, `reserve1`). In this case, the calculation for `getExchangeRate` will be performed incorrectly.

A similar method `getRateFromDex` is also implemented in the contract `ExchangeRateHelper`.

Recommendations

Add logic that takes into account the possibility of tokens with different `decimals` being in a pair.

[H-06] Using `tx.origin` creates an opportunity for phishing attack

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1314-L1321

Severity

Impact: High, as this will lead to a monetary loss for user

Likelihood: Medium, as the attack is not easy to execute

Description

The method `deposit` is implemented so that `transferFrom` contains `tx.origin` as parameter `from`.

```
    if ((DepositType(depositType) == DepositType.BY_EMP_ETH_LP)) {
      empEthLpToken.transferFrom(tx.origin, address(microgridNFTContract),
amount);

    } else if ((DepositType(depositType) == DepositType.DEFAULT)) {
      empToken.transferFrom(msg.sender, address(microgridNFTContract),
amount);

    } else if ((DepositType(depositType) == DepositType.BY_UPEMP)) {
      upEmp.transferFrom(tx.origin, address(microgridNFTContract),
amount);
    }
```

Using `tx.origin` is a dangerous practice as it opens the door to phishing attacks.

Recommendations

To prevent `tx.origin` phishing attacks, `msg.sender` should be used instead of `tx.origin`.

[H-07] Missing slippage checks, deadline check is not effective

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1338

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1374

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1340

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1378

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: Medium, as it is not expected to happen every time, but there are multiple attack paths here

Description

The `run` (and `runFromUpkeep`) make a dangerous assumption about `slippage`, namely that there is not any. The `deadline` check is set to `block.timestamp + 120`, which means the deadline check is disabled.

Users can be frontrun and receive a worse price than expected when they initially submitted the transaction. There's no protection at all, no minimum return amount or deadline for the trade transaction to be valid which means the trade can be delayed by miners or users congesting the network, as well as being sandwich attacked - ultimately leading to loss of user funds.

Functions with these issues:

- `WBNBBatteryInteract.run`
- `WBNBBatteryInteract.runFromUpkeep`
- `WETHBatteryInteract.run`
- `WETHBatteryInteract.runFromUpkeep`

Recommendations

Consider adding slippage protection, `amountOutMinimum` can be either set manually or calculated based on external oracles.

[H-08] Incorrect `path` array length specified

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1335-L1338

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1373-L1376

Severity

Impact: Medium, as no value will be lost but the contract functionality will be limited

Likelihood: High, as the functions will just revert every time

Description

The methods `run` and `runFromUpkeep` have incorrectly specified `path` arrays lengths.

Recommendations

Change the code in the following way:

```
-     address[] memory path = new address[](2);
+     address[] memory path = new address[](3);
      path[0] = address(eshareToken);
      path[1] = address(WBNB);
      path[2] = address(WETH);
```

[H-09] Storage mapping `lastClaimTime` is never initialized

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWETH.sol

Severity

Impact: Medium, as no value will be lost but the contract functionality will be limited

Likelihood: High, as it happens every time

Description

The storage mapping `lastClaimTime` is never initialized.

Recommendations

Add to methods `run` and `runFromUpkeep` the logic that will initialize `lastClaimTime`

[M-01] Additional check in `listClaimableReceivers` results in an empty list of recipients

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionAutoClaimUpkeep.sol#L32

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L428

Severity

Impact: Medium, as this will lead to a incomplete use of protocol capabilities

Likelihood: Medium, as it happens only for using of AutoClaim

Description

Method `checkUpkeep` generates a list of receivers using an external call to method `listClaimableReceivers` on contract `FusionRewardDistributor`. Since this happens before receiving rewards on `FusionRewardDistributor` through `_tryClaimFarm`, the method getTotalRewards gives a result of 0, and the receiver will not be included in the eligibleReceivers list (according to the following code):

```
if (getTotalRewards(_tokenId, _receiver) >=
IBattery(_receiver).minClaimAmount()) {
  eligibleReceivers[eligibleCount++] = _receiver;
}
```

Therefore checkUpkeep will generate always an empty list and method performUpkeep a will not work.

Recommendations

Remove check in the method listClaimableReceivers. Change the code in the following way:

```
  function listClaimableReceivers(uint256 _startIndex, uint256 _endIndex)
external view returns (uint256[] memory, address[][] memory) {
    // Ensure _endIndex is within the bounds of the active token IDs array
    _endIndex = Math.min(_endIndex + 1, activeTokenIds.length());

    // Initialize temporary arrays to hold token IDs and receivers
    uint256[] memory tokenIds = new uint256[](_endIndex - _startIndex);
    address[][] memory temp = new address[][](_endIndex - _startIndex);

    uint256 counter = 0; // A counter to keep track of how many eligible
receivers we've found

    // Loop over the specified range of token IDs
    for (uint256 i = _startIndex; i < _endIndex; i++) {
      uint256 _tokenId = activeTokenIds.at(i);
      uint256 _numReceivers = activeReceivers[_tokenId].length();

      // Initialize an array to hold the eligible receivers for the
current token ID
      address[] memory eligibleReceivers = new address[](_numReceivers);
      uint256 eligibleCount = 0;

      // Loop over all receivers for the current token ID
      for (uint256 j = 0; j < _numReceivers; j++) {
        address _receiver = activeReceivers[_tokenId].at(j);

-       // If the receiver can claim rewards, add them to the
eligibleReceivers array
-       if (getTotalRewards(_tokenId, _receiver) >=
IBattery(_receiver).minClaimAmount()) {
          eligibleReceivers[eligibleCount++] = _receiver;
-       }
      }

        ........................
```

[M-02] User may be incorrectly excluded from the list of `activeTokenIds`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L95

Severity

Impact: Medium, as it will not lead to the loss of rewards, but makes them unavailable through one method

Likelihood: Medium, as it occurs only when calling automation

Description

The method `_setReceiver` removes the user's `microgridNftId` from the list of `activeTokenIds` if `_allocPoints == 0` for `_receiver`. As it's implemented with this code:

```
    if (_allocPoints > 0) {
      if (!activeReceivers[_microgridNftId].contains(_receiver)) {
        activeReceivers[_microgridNftId].add(_receiver);
      }
      if (!activeTokenIds.contains(_microgridNftId)) {
        activeTokenIds.add(_microgridNftId);
      }
    } else {
      if (activeReceivers[_microgridNftId].contains(_receiver)) {
        activeReceivers[_microgridNftId].remove(_receiver);
      }
      if (activeTokenIds.contains(_microgridNftId)) {
        activeTokenIds.remove(_microgridNftId);
      }
    }
```

However, the user may still have other receivers with `_allocPoints > 0`. Excluding a user from the list of `activeTokenIds` will result in him being excluded from the distribution of rewards through a method `performUpkeep` in `FusionAutoClaimUpkeep`.

Recommendations

Add additional check that the user has no active receivers left. Change the code in the following way:

```
 - if (activeTokenIds.contains(_microgridNftId)) {
 + if (activeTokenIds.contains(_microgridNftId) &&
 activeReceivers[_microgridNftId].length() == 0) {
     activeTokenIds.remove(_microgridNftId);
   }
```

[M-03] Too large `batchSize` can lead to DoS

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionAutoClaimUpkeep.sol#L23

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L428

Severity

Impact: High, as the contract will be in a state of DoS, without a way for use `FusionAutoClaimUpkeep`

Likelihood: Low, as it requires a really big `batchSize`

Description

The `listClaimableReceivers` method in contract `FusionRewardDistributor` loop over the `activeTokenIds`. The number of iterations is determined by the parameter `batchSize` in the contract `FusionAutoClaimUpkeep`. If at some point there are now so large `batchSize`, iterating will become very costly and can result in a gas cost that is over the block gas limit. This would lead to a Denial of Service as users wouldn't be able to properly use the protocol functions.

Recommendations

Limit the `batchSize` that can be used to a value low enough that there is no risk of running out of gas.

[M-04] Missing checking that input array lengths are equal to each other

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L920

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryManager.sol#L64

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1156

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L990

Severity

Impact: Medium, as it will not lead to the loss of funds

Likelihood: Low, as it requires a big error on owner's side

Description

In contract `MarketplaceInteract`

- `setBonusTiers` method does not check if input array lengths are equal to each other.

In contract `MicrogridBatteryManager`

- `activateBattery` method does not check if input array lengths are equal to each other.

In contract `MicrogridNFTDeposit`

- `setAggregators` method does not check if input array lengths are equal to each other.

In contract `ExchangeRateHelper`

- `setAggregators` method does not check if input array lengths are equal to each other.

Recommendations

Add a checks that input array lengths are equal to each other.

[M-05] Missing checking on the setter method that the length of input array is equal to the required value

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L910C1-L925C4

Severity

Impact: High, as it will result in block the functionality of contract

Likelihood: Low, as it requires a big error on owner's side

Description

The `getBonusPercent` method loop over the `bonusTiers` array with condition `i < 10`.

```
  function getBonusPercent(uint256 amount) internal view returns (uint256)
{
    for (uint256 i = 0; i < 10; i++) {
      if (amount >= bonusTiers[i].amount) {
        return bonusTiers[i].percent;
      }
    }

    return 0;
  }
```

However, in the method `setBonusTiers` the length of the `bonusTiers` array is not limited in any way. Thus, if the length of the `bonusTiers` array will be less than 10, then calling `getBonusPercent` will result in revert. This will block the functionality of the method `buyPoints.`

Recommendations

Add a check that input array lengths in the method `setBonusTiers` are equal to 10.

[M-06] Missing checking on the setter method of a descending order of values

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L910C1-L930

Severity

Impact: High, as it will result in wrong points calculations

Likelihood: Low, as it requires a big error on owner's side

Description

The logic of the method `getBonusPercent` involves searching through `bonusTiers[i].amount` values, starting from largest to smallest (in order to find the first “greater than”). This imposes special requirements on the formation of the order of values within the array passed to the setter `setBonusTiers`. However, the method `setBonusTiers` does not check array values. This can lead to `amount` values in the `bonusTiers` array not being in descending order, which will lead to incorrect calculation of bonusPercent.

Also, this logic must be taken into account when making changes to a separate `BonusTier` through the method `setBonusTier`.

Recommendations

Add a check in the method `setBonusTiers` that `amount` values in the `bonusTiers` input array are in descending order. Add a check in the method `setBonusTier` that `amount` value with `tierIndex` is less than the previous one (`tierIndex - 1`) and greater than the next one (`tierIndex + 1`).

[M-07] Oracle price is used without checking validity

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L932

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1168

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L940

Severity

Impact: Medium, it affects user assets only when the price feed oracle is in bad status

Likelihood: Medium, it affects only when the price feed oracle is in bad status

Description

Some methods fetches data from Chainlink (or another price feed) with `IAggregator` and `latestAnswer`, `latestRoundData`. To ensure accurate price usage, it's vital to regularly check the last update timestamp against a predefined delay. However, the current implementation lacks checks for the staleness of the price obtained from price feed. Without proper checks, consumers of protocol may continue using outdated, stale, or incorrect data if oracles are unable to submit and start a new round.

Functions using price feeds

- `MarketplaceInteract.buyPoints`
- `MicrogridNFTDeposit.getRateFromChainlink`
- `ExchangeRateHelper.getRateFromChainlink`

Recommendations

Implement a mechanism to check the heartbeat of the price feed and compare it against a predefined maximum delay (`MAX_DELAY`). Adjust the `MAX_DELAY` variable based on the observed heartbeat. It is recommended to implement checks to ensure that the price returned by price feed is not stale.

[M-08] Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L932

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1255

Severity

Impact: Medium, it affects user assets only with tokens that don’t correctly implement the latest EIP20 spec

Likelihood: Medium, it affects only with tokens that don’t correctly implement the latest EIP20 spec

Description

There is `transferFrom` calls that do not check the return value (some tokens signal failure by returning false). It is a good idea to add a require() statement that checks the return value of ERC20 token transfers or to use something like OpenZeppelin’s safeTransfer()/safeTransferFrom() unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in contract.

However, using require() to check transfer return values could lead to issues with non-compliant ERC20 tokens which do not return a boolean value. Therefore, it's highly advised to use OpenZeppelin’s safeTransfer()/safeTransferFrom().

Proof of Concept

MarketplaceInteract.sol

L945: `IERC20(token).transferFrom(msg.sender, address(marketplaceContract), amount);`

L1326: `(IERC20(currency)).transferFrom(msg.sender, address(microgridNFTContract), amount);;`

Recommendations

Consider using `safeTransfer()/safeTransferFrom()` instead of `transfer()/transferFrom()`.

[M-09] Wrong use of `transferFrom` ERC721A for transfer to `address(0)`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPosition.sol#L232

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWBNB.sol#L225

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWETH.sol#L225

Severity

Impact: Medium, as it will not lead to the loss of funds

Likelihood: Medium, as it does not affect the functionality of the entire protocol

Description

The `upgradeBattery` removes ownership of battery and then must `burn BatteryNFT`. The issues are that:

- contract MicrogridBatterySplitMyPosition (MicrogridBatteryWBNB, MicrogridBatteryWETH) inherits from contract ERC721A, in which transferFrom to the address(0) is prohibited
- usersMicrogridToken Id is passed as a parameter for burn, but the MicrogridBatterySplitMyPosition Id (MicrogridBatteryWBNB Id, MicrogridBatteryWETH Id) should be passed (in addition, it's necessary to take into account that all MicrogridBatterySplitMyPosition NFT (MicrogridBatteryWBNB NFT, MicrogridBatteryWETH NFT) belong to the address(this))
- in the current implementation the method does not work

Recommendations

It is recommended to use the method _burn with the correct NFT Id as parameter.

[M-10] Missing Min Max validation for sacMultiplier

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1093

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: Low, as it requires a big error on owner's side

Description

The method setSacrificeContract is implemented so that there are no any restrictions on the minimum and maximum values of the parameter sacMultiplier. This parameter is used in calculating the size of the user’s deposit in method deposit (in particular, sacMultiplier == 0 will result in the complete loss of the user's funds).

Recommendations

It is recommended to add storage variables defining Min and Max values of sacMultiplier. Add a suitable check to the method setSacrificeContract.

[M-11] Lack of price validation in the method setManualPrice

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1151

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: Low, as it requires a malicious/compromised owner or a big error on his side

Description

The `setManualPrice` method allows `owner` to set a price without any validation. This approach significantly increases the risk of protocol centralization.

Recommendations

Add validation for owner set price using external oracles.

[M-12] Missing checks for user's `Microgrid NFT` ownership of `battery`, `treasury` ownership of `Microgrid NFT`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L240

Severity

Impact: High, as it will lead to the loss of funds and restriction on the functionality of protocol

Likelihood: Low, as it occurs only in this method

Description

The method `split` is implemented so that it missed checks that:

- user's Microgrid NFT must own the battery
- the treasury must own Microgrid NFT

This may result in the method `split` being used by a user who does not own the `battery`, causing financial loss to the protocol.

Recommendations

Change the code in the following way:

```
  function split(uint256 sharesAmount, uint256 pricePerShare) public
nonReentrant {
    // Find user's microgrid token ID.
    uint256 usersMicrogridToken =
microgridContract.tokenByWallet(msg.sender);
```

```
    // Requires
    require(usersMicrogridToken > 0, "You must own a Microgrid NFT.");
+   require(batteryContract.checkBattery(usersMicrogridToken) == true,
"Your Microgrid does not own this battery.");
    require(sharesAmount <=
microgridContract.individualShares(usersMicrogridToken), "You cannot split
more shares than you own.");

    // Remove the sharesAmount from user's individualShares temporarily.
    uint256 currentShares =
microgridContract.individualShares(usersMicrogridToken);
    uint256 newShares = currentShares - sharesAmount;
    microgridContract.setShares(newShares, usersMicrogridToken);

    // Add the sharesAmount temporarily to the Treasury wallet.
    uint256 treasuryMicrogridToken =
microgridContract.tokenByWallet(treasury);
+   require(treasuryMicrogridToken > 0, "Treasury must own a Microgrid
NFT.");
    uint256 treasuryCurrentShares =
microgridContract.individualShares(treasuryMicrogridToken);
    uint256 treasuryNewShares = treasuryCurrentShares + sharesAmount;
    microgridContract.setShares(treasuryNewShares,
treasuryMicrogridToken);
    ....................
```

[M-13] Use of `transfer` might render BNB impossible to withdraw

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L344-L345

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L356

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1332

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L971

Severity

Impact: High, as it will lead to the loss of funds and restriction on the functionality of protocol

Likelihood: Low, as it occurs only in this method

Description

When withdrawing BNB, the `BatteryInteractSplitMyPosition`, `MicrogridNFTDeposit`, `MarketplaceInteract` contracts use Solidity's `transfer` function. This has some notable shortcomings when the withdrawer is a smart contract, which can render BNB impossible to withdraw. Specifically, the withdrawal will inevitably fail when:

- The withdrawer smart contract does not implement a payable fallback function.
- The withdrawer smart contract implements a payable fallback function which uses more than 2300 gas units.
- The withdrawer smart contract implements a payable fallback function which needs less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

Recommendations

Recommendation is to stop using `transfer` in code and switch to using `call` instead. Additionally, note that the `sendValue` function available in OpenZeppelin Contract's `Address` library can be used to transfer the withdrawn BNB without being limited to 2300 gas units.

[M-14] Missing Min Max validation for `autoCompoundMultiplier`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol

Severity

Impact: High, as this will lead to a monetary loss for users

Likelihood: Low, as it requires a big error on owner's side

Description

The methods `setCompoundContract`, `updateAutoCompoundMultiplier` are implemented so that there are no any restrictions on the minimum and maximum values of the parameter `autoCompoundMultiplier`. This parameter is used in calculating the size of the user's `compoundAmount` in the methods `compoundForTokenId`, `compoundForUser` (in particular, `autoCompoundMultiplier == 0` will result in the complete loss of the user's funds).

Recommendations

It is recommended to add storage variables defining Min and Max values of `autoCompoundMultiplier`. Add a suitable check to the methods `setCompoundContract`, `updateAutoCompoundMultiplier`.

[L-01] Disabled `microgridBatteryNFTContracts` are not removed from the array

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryManager.sol#L59

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryManager.sol#L64

Severity

Impact: Low, as it will not lead to the loss of funds or restrict the functionality of contract

Likelihood: Low, as it happens only for the case of removing battery

Description

The method `activateBattery` when disabling a previously added `microgridBatteryNFTContracts` (`allocPoints == 0`), does not remove it from the array `batteryInfo.ownedBatteries`. Therefore, the method `getBatteryListByUser` will always return a complete list of all user's `microgridBatteryNFTContracts` (with `allocPoints == 0` and `allocPoints > 0`).

Recommendations

Add logic that removes inactive `microgridBatteryNFTContracts` from the array (using OpenZeppelin's EnumerableSet is also possible).

[L-02] Missing zero address checks

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1031

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1156

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L273

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L310

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L314

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L322

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L326

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFT.sol#L336

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L409

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1429 https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1433

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L881

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/FusionRewardDistributor.sol#L87

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L1018

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPosition.sol#L150

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPosition.sol#L177

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWBNB.sol#L150

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWBNB.sol#L177

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWETH.sol#L150

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWETH.sol#L177

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/OldFusionRewardDistributor.sol#L88

Severity

Impact: Low, as it will not lead to the loss of funds or big restriction on the functionality of protocol

Likelihood: Low, as it requires a big error on owner's side

Description

Contracts have address fields in multiple methods. These methods are missing address validations. Each address should be validated and checked to be non-zero. This is also considered a best practice. A wrong user input or defaulting to the zero addresses for a missing input can lead to the contract needing to redeploy or wasted gas.

Functions with missing zero address checks

- `MicrogridNFTDeposit.constructor`
- `MicrogridNFTDeposit.setAggregators`
- `MicrogridNFT.allowMultiple`
- `MicrogridNFT.setDepositContract`

- `MicrogridNFT.setRewardContract`
- `MicrogridNFT.setWhitelistedContracts`
- `MicrogridNFT.setCompoundContract`
- `MicrogridNFT.setSigner`
- `BatteryInteractSplitMyPosition.setTreasury`
- `WBNBBatteryInteract.setUpkeepContract`
- `WETHBatteryInteract.setUpkeepContract`
- `ExchangeRateHelper.constructor`
- `FusionRewardDistributor.setBatteryManagerContract`
- `MarketplaceInteract.setWrapped`
- `MicrogridBatterySplitMyPosition.setInteractContract`
- `MicrogridBatterySplitMyPosition.setUpgradeBatteryContract`
- `MicrogridBatteryWBNB.setInteractContract`
- `MicrogridBatteryWBNB.setUpgradeBatteryContract`
- `MicrogridBatteryWETH.setInteractContract`
- `MicrogridBatteryWETH.setUpgradeBatteryContract`
- `OldFusionRewardDistributor.setBatteryManagerContract`

Recommendations

It is recommended to validate that each address input is non-zero.

[L-03] Missing validation for `bonusStartTime` and `bonusEndTime` of `BonusMultiplier`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1062

Severity

Impact: Low, as it will not lead to the loss of funds or big restriction on the functionality of protocol

Likelihood: Low, as it requires a big error on owner's side

Description

The method `setBonusMultiplier` do not validate that `bonusStartTime` is less than `bonusEndTime`. If those values are not correctly set, then applying the parameter `bonusMultiplier` will become impossible.

Recommendations

Add a validation logic inside `setBonusMultiplier` method to ensure that `bonusStartTime` is less than `bonusEndTime`.

[L-04] Missing setting of `token` for `allowedCurrencies`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1072

Severity

Impact: Low, as it will not lead to the loss of funds or restriction on the functionality of protocol

Likelihood: Low, as it occurs only in this method

Description

The method `setAllowedCurrency` is implemented so that it missed the setting of `allowedCurrencies[_token].token`.

Recommendations

Change the code in the following way:

```
  function setAllowedCurrency(
    address _token,
    address _lpToken,
    uint256 _sharesPerEMP,
    bool _allowed
  ) public onlyOwner {
+   allowedCurrencies[_token].token = _token;
    allowedCurrencies[_token].lpToken = _lpToken; // The Cake-LP token for
the primary trading pair for this token.
    allowedCurrencies[_token].sharesPerEMP = _sharesPerEMP; // Amount of
microgrid shares allocated per $1 deposited.
    allowedCurrencies[_token].allowed = _allowed; // Only allowed = true
tokens are able to be deposited.
  }
```

[I-01] Unused code

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L699

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L703 https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L703

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L119

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1106

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1106

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MarketplaceInteract.sol#L879

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1238

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1238

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1247

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1247

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1249

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1249

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1255

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1255

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1263

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1263-L1265

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1264-L1266

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L896

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L896 https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWBNB.sol#L106

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatteryWETH.sol#L106

Description

The `AggregatorV3Interface` interface in `MarketplaceInteract` is not used anywhere and should be removed.

The `AggregatorV3Interface` interface in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `AggregatorV3Interface` interface in `WETHBatteryInteract` is not used anywhere and should be removed.

The `AggregatorV3Interface` interface in `MicrogridBatterySplitMyPositionInteract` is not used anywhere and should be removed.

The `IPancakeFactory` interface in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `IPancakeFactory` interface in `WETHBatteryInteract` is not used anywhere and should be removed.

The `PriceRecord` struct in `MarketplaceInteract` is not used anywhere and should be removed.

The `allowedCurrencyInfo` struct in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `allowedCurrencyInfo` struct in `WETHBatteryInteract` is not used anywhere and should be removed.

The `pancakePairToken` storage variable in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `pancakePairToken` storage variable in `WETHBatteryInteract` is not used anywhere and should be removed.

The `empToken` storage variable in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `empToken` storage variable in `WETHBatteryInteract` is not used anywhere and should be removed.

The `wrapped` storage variable in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `wrapped` storage variable in `WETHBatteryInteract` is not used anywhere and should be removed.

The `allowedCurrencies` storage mapping in `WETHBatteryInteract` is not used anywhere and should be removed.

The `allowedMultiple` storage mapping in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `allowedMultiple` storage mapping in `WETHBatteryInteract` is not used anywhere and should be removed.

The `ownsBattery` storage mapping in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `ownsBattery` storage mapping in `WETHBatteryInteract` is not used anywhere and should be removed.

The `batteriesActive` storage mapping in `WBNBBatteryInteract` is not used anywhere and should be removed.

The `batteriesActive` storage mapping in `WETHBatteryInteract` is not used anywhere and should be removed.

The `setReceiver` function in `IFusionRewardDistributor` is not used anywhere and should be removed.

The `revealed` storage variable in `MicrogridBatteryWBNB` is not used anywhere and should be removed.

The `revealed` storage variable in `WETHBatteryInteract` is not used anywhere and should be removed.

Recommendations

It is recommended to remove any unused code or provide valid recommendations and suggestions on the documentation on how to use those.

[I-02] Unused method `burn`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPosition.sol#L225

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol

Description

The method `burn` can only be externally called by the contract `MicrogridBatterySplitMyPositionInteract`.

```
  require(interactContract == msg.sender, "Only the interact contract can
burn.");
```

However, `MicrogridBatterySplitMyPositionInteract` implementation does not have the functionality to call method `burn` in `MicrogridBatterySplitMyPosition`.

Recommendations

Add the necessary functionality to the contract `MicrogridBatterySplitMyPositionInteract` or remove unused method `burn`.

[I-03] Adding a returns value `orderId`

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridBatterySplitMyPositionInteract.sol#L240

Description

For convenience of working with the protocol, recommend adding a returns value `orderId` to the method `split`.

[I-04] Ineffective TWAP implementation

The contracts `MicrogridNFTDeposit` and `MarketplaceInteract` uses prices based on `twap` from oracle. Despite the fact that the Oracle contract (0x0Fe57361B0E3Fc7F61972BD839Ddaa8Da3E691D2) is out of scope, we would like to draw attention to the fact that the `twap` was not implemented effectively enough. In particular, the `twap` window and order of `observations` are not defined.

[I-05] State variables could be declared immutable

Relevant GitHub Links

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WBNBBatteryInteract.sol#L1248

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/WETHBatteryInteract.sol#L1248

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/ExchangeRateHelper.sol#L856

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L998-L1000

https://github.com/DeFi-Gang/emp-fusion-contracts/blob/main/contracts/fusion/MicrogridNFTDeposit.sol#L1022

Description

State variables that are not updated following deployment should be declared immutable to save gas.

The `pancakeRouter` storage variable in `WBNBBatteryInteract` should be immutable.

The `pancakeRouter` storage variable in `WETHBatteryInteract` should be immutable.

The `wrapped` storage variable in `ExchangeRateHelper` should be immutable.

The `empEthLpToken` storage variable in `MicrogridNFTDeposit` should be immutable.

The `empToken` storage variable in `MicrogridNFTDeposit` should be immutable.

The `microgridNFTContract` storage variable in `MicrogridNFTDeposit` should be immutable.

The `wrapped` storage variable in `MicrogridNFTDeposit` should be immutable.