shieldify

PEAR PROTOCOL

SECURITY REVIEW

Date: 24 February 2024

CONTENTS

1. About Shieldify

We are Shieldify Security – Revolutionizing Web3 security. Elevating standards with top-tier reports and a unique subscription-based auditing model.
Learn more about us at shieldify.org or @ShieldifySec.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Pear Protocol

Pear Protocol represents an innovative solution designed to streamline and enhance the efficiency of on-chain pairs trading. It enables users to execute leveraged long and short positions within a single transaction, addressing the complexities and inefficiencies traditionally associated with pair trading in cryptocurrencies. By integrating a variety of on-chain trading engines alongside a dedicated user interface and experience, Pear Protocol simplifies the process of initiating simultaneous long and short positions in correlated assets, such as going long on BTC while shorting ETH with leverage.

This liquidity-agnostic platform offers deep liquidity access, and flexibility in managing trading parameters, and mitigates the custody and trust issues found in centralized exchanges by ensuring traders retain asset custody. Beyond simplifying trading executions, Pear Protocol extends its utility with a tokenized trading system, allowing trading positions to be represented as ERC-721 tokens for increased composability within DeFi ecosystems. With features emphasizing simplicity, flexibility, scalability, optionality, and ease of use, Pear Protocol aims to revolutionize pair trading, making it more accessible and efficient while fostering broader decentralized finance trading solutions adoption.

Learn more about Pear's concept and the technicalities behind it here.

3.1 Observations

- While Pear plans to integrate with multiple trading engines, like GMX, Vertex, and SYMM, this security review is concerned mostly with the GMX part. The only other engine used during this review was Vertex, however only part of its functionality was present.
- The main components of the system are GMXFactory and GMXAdapter, responsible for communication with GMX and position management; PlatformLogic managing protocol fees; PositionNFT tokenizing position into transferable NFT and Comptroller, serving as a singleton reference for all other contracts to find addresses of the system components.

3.2 Privileged Roles and Actors

- `GMXFactory.sol` – creates new pair positions (adapters) and implements functionalities to manage them.
- `GMXAdapter.sol` – GMX pair position adapter. An interface between GMX positions and position owners.

- `NFTPosition.sol` – allows for position tokenization and making it transferrable.
- `Admin` – has powers to set all the Pear protocol fees, and referrals and upgrade the protocol
- `Position owner` – user of the protocol. Are owners of GMXAdapter owning GMX position, hence owners of the positions.
- `Stakers and Treasury` – fees recipients.

4. Risk Classification

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

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security assessment spanned two weeks, during which the Shieldify team collectively dedicated 552 hours. The code incorporates fundamental best practices for security. Shieldify expresses their gratitude for Pear team's prompt and efficient communication, which greatly enhanced the quality of the audit report.

5.1 Protocol Summary

Project Name	**Pear Protocol**
Repository	pear-sc
Type of Project	DEX, On-Chain Pairs Trading
Audit Timeline	14 days
Review Commit Hash	00aa0ce64fd6fd95c3207e01871bdbe3267db2c9
Fixes Review Commit Hash	4bcb9bad4001fe1443e058b788e8d72382fae2d2

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/interfaces/IPositionRouter.sol	3
src/interfaces/IAcceptComptroller.sol	3
src/interfaces/IPearBase.sol	57
src/interfaces/IPositionRouterCallbackReceiver.sol	3
src/interfaces/IPlatformLogic.sol	87
src/interfaces/INftHandler.sol	30
src/interfaces/IPearGmxAdapter.sol	44
src/interfaces/IRouter.sol	3
src/interfaces/IVaultPriceFeed.sol	3
src/interfaces/IComptroller.sol	24
src/interfaces/IPositionNFT.sol	9
src/interfaces/IVertexFactory.sol	11
src/interfaces/IReader.sol	3
src/interfaces/IVaultUtils.sol	3
src/interfaces/IVault.sol	4
src/interfaces/IPearGmxFactory.sol	66
src/Factory/GmxFactory.sol	821
src/Factory/VertexFactory.sol	53
src/PlatformLogic.sol	669
src/libraries/Events.sol	58
src/libraries/Errors.sol	22
src/NFT/NftHandler.sol	124
src/NFT/PositionNFT.sol	50
src/GmxAdapter.sol	135
src/Comptroller.sol	149
Total	**2434**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **5**
- **Medium** issues: **13**
- **Low** issues: **11**

ID	Title	Severity
[H-01]	User May Pay Smaller Fees, Get No Refund, Or Force Treasury and Stakers to Not Receive Fees at All	High
[H-02]	The `_applyPlatformFeeCloseErc20Gmx()` Function Misses to Implement `GmxFactory.tokenTransferPlatformLogic()`	High
[H-03]	Insufficient Short Path Validation Locks Fees in `PlatfromLogic.sol` and Prevents Any State Change of a Position	High
[H-04]	The `withdrawClosedSize()` Function Misses the Situation Where the Funds Are Transferred as `ETH` From `GMX`	High
[H-05]	Vertex Fees Transferred From User Do Not Account for Referee Fees, Leaving The Protocol with Permanent Shortfall	High
[M-01]	Current Fee Structure May Harm the Users and Allows for Fee Exploitation	Medium
[M-02]	Transferring the Position NFT While There Is an Unexecuted Position Might Lose the Fees of The New Adapter Owner	Medium
[M-03]	The Referrer Receives the Fees Even If the Position Execution Fails	Medium
[M-04]	The Protocol Fails to Create a Unique Salt in Some Conditions	Medium
[M-05]	NFT Transfer Disallowed When Any of the Side Is Changed to Zero	Medium
[M-06]	Fees Sent to `PlatformLogic.sol` Contract While Closing Position Do Not Account For Insufficient Adapter Funds	Medium
[M-07]	Reusing the Same `minOut` For Long and Short Position Makes Slippage Protection Ineffective, and May Result in DoS for Opening New Positions	Medium
[M-08]	No Slippage Protection for `Decrease` and `Close` Position May Create a Trade That is Unfavourable to The User	Medium
[M-09]	`PositionNft` Cannot Be Transferred By Itself	Medium
[M-10]	The Old Position Owner Could Cause Damage to The New Position Owner	Medium
[M-11]	If NFT Is Transferred to Current Owner, It Is Lost	Medium
[M-12]	Introduction of Global Referrer Fees Leads to Unexpected Reverts And Inability to Create Certain Positions	Medium
[M-13]	Division Before Multiplication at GmxFactory's `feeSplit` Calculation	Medium
[L-01]	Users Are Unable to Deleverage Their Positions	Low
[L-02]	Position State Can Be Broken If Referee Is Blacklisted	Low

ID	Title	Severity
[L-03]	The Next New Position Index of The Old Owner Could Overwrite The Existing Indexes of The Old Owner	Low
[L-04]	Missing Sanity Checks in `setTreasuryFeeSplit()` Function, Allows to Apply Over 100 Percent Fees	Low
[L-05]	Admin Can Make a `Position NFT` Unmineable by `Blacklisting` Position Collateral Token	Low
[L-06]	Mint Fee is Not Aligned With the Docs and Also Big Fees Can be Avoided	Low
[L-07]	No Possibility to Close Liquidated Trade	Low
[L-08]	Functions Need To Be Called Only Once	Low
[L-09]	The `CalculateFees()` Function Will Revert If Any Low Decimal Token Is Allowed as Fee Token	Low
[L-10]	Centralization Risks	Low
[L-11]	`FeeSplit` Calculation Might Truncate to Zero	Low

7. Findings

[H-01] User May Pay Smaller Fees, Get No Refund, Or Force Treasury and Stakers to Not Receive Fees at All

Severity

High Risk

Description

While an issue with fee structure is mentioned in `Current fee structure may harm the users, and allows for fee exploitation`, there is a way more severe issue concerning fees, which involves overriding pending trades with new orders. The `setRefereeFeeAmount()` function is called whenever there's an order for `increase/decrease` calls:

File: src/PlatformLogic.sol#L239

```
/// @inheritdoc IPlatformLogic
function setRefereeFeeAmount(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount
) public override onlyFactory {
  refereeFeeAmounts[_referee][_adapterAddress] = _feeAmount;
  emit SetRefereeFeeAmount(_referee, _adapterAddress, _feeAmount);
}
```

So if a user opens a position -> it will be called with a `0,25` fee. If the same user increases the position without the position being executed -> it's overwritten with the `0,1` fee.

And if the first order fails, the `handleTokenAmountWhenPositionHasFailed()` function to be called inside `_handleFeeAmount`:

File: src/PlatformLogic.sol#L287

```
/// @inheritdoc IPlatformLogic
function handleTokenAmountWhenPositionHasFailed(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount,
  IERC20 _tokenAddress
) external override onlyFactory {
  if (refereeFeeAmounts[_referee][_adapterAddress] == 0) {
    revert PlatformLogic_FeeAmountCannotBeNull();
  }
  setRefereeFeeAmount(_referee, _adapterAddress, 0);

  SafeTransferLib.safeTransfer(
    address(_tokenAddress),
    _referee,
    _feeAmount
  );
}
```

And since the `_feeAmount` is greater than the latest fee, the funds will get stuck:

File: src/Factory/GmxFactory.sol#L532-L549

```
/// @inheritdoc IPearGmxFactory
function gmxPositionCallback(
  bytes32 positionKey,
  bool isExecuted,
  bool isIncrease
) external override {
// code

// If both long/short fails either on increase or decrease
if (
  longExecutionState == ExecutionState.Failed &&
  (shortExecutionState == ExecutionState.Failed)
) {
  IPearGmxAdapter(adapter).closeFailedPosition(
    pairData.short.path,
    adapterOwner
  );
```

```
 _handleFeeAmount(
   adapterOwner,
   adapter,
   platformLogic,
   collateralToken,
   feeAmount,
   true
 );
} else if (

// code
}
```

The same mechanism can be used to pay only half of the fees by the user. When a position is increased or decreased, the user has to pay the fee based on the current position size in GMX. In case the two-sided position fails, the user gets a **100 percent** refund of `refereeFeesAmount`. If only one side fails, the user is refunded **50 percent** in their lines:

```
IPearGmxAdapter(adapter).closeFailedPosition(pairData.short.path,
   adapterOwner);

IPlatformLogic(platformLogic).setRefereeFeeAmount(
  adapterOwner,
  adapter,
  feeAmount / 2
);

_handleFeeAmount(
  adapterOwner,
  adapter,
  platformLogic,
  collateralToken,
  feeAmount / 2,
  false
);
_handleFeeAmount(
  adapterOwner,
  adapter,
  platformLogic,
  collateralToken,
  feeAmount / 2,
  true // @audit Failed flag. Will refund feeAmount / 2 to the user
);
```

So, if the user wants to create a single-sided order to change only one leg, he can create an opposite side order with minuscule amounts and faulty params that will fail in GMX, like swap path longer than 2, which is checked by GMX and not by Pear. This way second order will fail and half of the fees will be refunded to the user.

An additional problem is caused by the fact that with each increase and decrease operation, `refereeFeeAmount` is overridden, which means that if a position is changed twice before is

executed, the user and/or treasury and stakers won't get their funds back.

Impact

1. Fund Mismanagement: Incorrect fee handling can lead to funds being stuck or improperly refunded.
2. Exploitation for Reduced Fees: Users can exploit this issue to pay significantly reduced fees by creating faulty operations.
3. The user may decide to force the fee assets to be locked, without the possibility of retrieving them by

Attack Scenario

Losing fees

1. The user creates a big position, creates an increase and decides that he wants to mint an NFT.
2. Minting NFT overrides `refereeFeeAmount`.
3. Because the increased transaction failed in GMX, the user gets a refund, but for the amount that was for an NFT.

This is a complex scenario, however, an overriding fee happens each time that position is increased, decreased, refunded after failure, or minting NFT.

Paying 50% of fees

1. The user wants to update one leg of his position, paying only 50% of the costs. The easy option is to create his target position and another one that will be guaranteed to fail, by swapping path that has 3 elements, which will fail in GMX.
2. The user's target lets the order succeed, and GMX fallback logic refunds him 50% of the fees, while he should pay 100% for it.

Locking treasury fees

1. The user has a small position already in GMX - $10.
2. User creates 3 new increase positions a position that will fail. The user pays fees for his current position size in GMX (for $10). They wait for the keeper to execute.
3. User passes 1 wei to his GmxAdapter. It will be required in the next step.
4. The first one is executed and in the callback, the user reenters via returning ETH to the user, and creates an increased position. Fee will be taken from the user, however when the reentrancy ends, `refereeFeeAmount` will be reset to 0.
5. When the increase operation is executed, it will revert, because of the check in GMX callback. The user won't get anything from this attack, but the treasury and stakers won't get their due money.

Location of Affected Code

File: src/PlatformLogic.sol#L239-L246

File: src/PlatformLogic.sol#L287-L303

File: src/Factory/GmxFactory.sol#L604-L625

Recommendation

1. Only allow one pending operation per adapter at a time.
2. Add reentrancy guards into the system to protect against this attack vector.
3. Use constant fee for minting NFT as described in the documentation and abstain from overriding `refereeFeeAmount` in this case.

Team Response

Acknowledged and fixed as suggested.

[H-02] The `_applyPlatformFeeCloseErc20Gmx()` Function Misses to Implement `GmxFactory.tokenTransferPlatformLogic()`

Severity

High Risk

Description

The `GmxFactory.closePositions()` function creates a position order to close an existing position. The platform fees also are included in this order at **`Line:396`** as below:

File: src/Factory/GmxFactory.sol#L396-L401

```
/// inheritdoc IPearGmxFactory
function closePositions(ClosePositionsArgs memory args) external payable
    override returns (
  address adapter,
  bytes32 longPositionId,
  bytes32 shortPositionId
) {
// code
>  uint256 feeAmount = IPlatformLogic(platformLogic)
    .applyPlatformFeeCloseErc20Gmx(
      msg.sender,
      sizeAmount,
      IERC20(pairPositionData[args.adapter].short.collateralToken)
    );
// code
}
```

`IPlatformLogic(platformLogic).applyPlatformFeeCloseErc20Gmx()` is called to take the platform fee and it calls `_applyPlatformFeeCloseErc20Gmx()` to implement here.

While this function updates the token amounts for the referrer withdrawal between lines **#530-#534** by calling the `addTokenFeesForWithdrawal()` function:

File: src/PlatformLogic.sol#L305-L314

```
function addTokenFeesForWithdrawal(
  address _referrer,
  uint256 _amount,
  IERC20 _token
) public override notZeroAddress(_referrer) onlyFactory {
// add a withdrawTokenFee function
  pendingTokenWithdrawals[_referrer][_token] += _amount;
  emit PendingTokenWithdrawal(_referrer, _amount);
}
```

It doesn't call the `GmxFactory.tokenTransferPlatformLogic()` function to transfer the referee fees to the `GmxFactory.sol` contract. So the referrer's `pendingTokenWithdrawals` mapping will be incremented by the re-calculated fee amount but the contract will not hold the `feeAmount`.

This could be abused by an actor who both holds the referee and the referral accounts. Accordingly, they can inflate their `pendingTokenWithdrawals`, call `withdrawTokenFees()` and drain the contract;

File: src/PlatformLogic.sol#L316-L325

```
function withdrawTokenFees(IERC20 _token) public override nonReentrant {
  uint256 _balance = pendingTokenWithdrawals[msg.sender][_token];
  if (_balance == 0) revert PlatformLogic_NotEnoughBalance();
  pendingTokenWithdrawals[msg.sender][_token] = 0;

  SafeTransferLib.safeTransfer(address(_token), msg.sender, _balance);

  emit TokenWithdrawal(msg.sender, _balance);
}
```

Impact

The `PlatformLogic.sol` contract could be drained, pending positions will not have tokens to pay for fees and will revert on GMX callback.

Location of Affected Code

File: src/PlatformLogic.sol#L495-#L553

Recommendation

`GmxFactory.tokenTransferPlatformLogic()` should be implemented in the `_applyPlatformFeeCloseErc20Gmx()` function.

Team Response

Acknowledged and fixed as suggested.

[H-03] Insufficient Short Path Validation Locks Fees in `PlatfromLogic.sol` and Prevents Any State Change of a Position

Severity

High Risk

Description

When creating a new position, Pear checks if the tokens are expected:

File: src/Factory/GmxFactory.sol#L96-L104

```
if (args.pathLong[0] != args.pathShort[0]) {
  revert Errors.GmxFactory_DifferentCollateralToken();
}
if (args.indexTokenLong == args.indexTokenShort) {
  revert Errors.GmxFactory_SameIndexToken();
}
if (comptroller.allowedTokens(IERC20(args.pathLong[0])) != true) {
  revert Errors.GmxFactory_TokenNotAllowed();
}
```

However, after creation, when new position data are set, there is a critical bug:

File: src/Factory/GmxFactory.sol#L1065

```
function setPositionData(
  address adapter,
  address[] memory path,
  address indexToken,
  bytes32 positionId,
  bool isLong,
  bool isIncrease
) internal {
  PositionData memory data;
```

```
  if (isIncrease) {
    data = PositionData({
      isLong: isLong,
      path: path,
// @audit The following line is invalid, as this is the destination token
    , not the token utilized by the individual
      collateralToken: path[path.length - 1],
      indexToken: indexToken,
      positionId: positionId,
      increaseExecutionState: ExecutionState.Pending,
      decreaseExecutionState: ExecutionState.Idle
    });
  } else {
    data = PositionData({
      isLong: isLong,
      path: path,
// @audit The following line is invalid, as this is the destination token
    , not the token utilized by the individual:
      collateralToken: path[path.length - 1],
      indexToken: indexToken,
      positionId: positionId,
      increaseExecutionState: ExecutionState.Idle,
      decreaseExecutionState: ExecutionState.Pending
    });
  }

  if (isLong) {
    data.collateralToken = indexToken;
    pairPositionData[adapter].long = data;
  } else {
    pairPositionData[adapter].short = data;
  }
}
```

The `collateralToken` is set to the last path token. This breaks `gmxPositionCallback()` function, which takes the token to pay fees with from short token collateral, and reverts, because the contract doesn't have the assets:

File: src/Factory/GmxFactory.sol#L477

```
function gmxPositionCallback(
  bytes32 positionKey,
  bool isExecuted,
  bool isIncrease
) external override {

// code
// @audit This will be set to invalid token if a short token swap path is
    used
  address collateralToken = pairData.short.collateralToken;
```

```
// code

  _handleFeeAmount(
    adapterOwner,
    adapter,
    platformLogic,
// @audit This will revert if collateralToken is not a token that the
   user paid with
    collateralToken,
    feeAmount,
    true
  );
// code
}
```

This makes the state to not be changed. Hence position state is not set to `Opened`, meaning that the user cannot increase, decrease or close the position. This makes users lose 100% of the funds in this case, as the position is unrecoverable. Additionally, all the fees that should be distributed, are locked in the contract.

Impact

If a user passes a short swap path that is valid from a GMX perspective, the user will lose his assets and the protocol will lose fees.

Proof of Concept (PoC)

The following test fails on the assertions at the end. When the test logs are viewed, it can be seen, that the GMX callback of the opening position fails, hence the position status is not set to `Opened`:

```
function test_OpenPositions_BothPositionsOpenningUSDCtoDAI() public {
// Amount in USDT
  uint256 pearFee = 250000;
// Assuming 15 USDC
  uint256 amountIn = 15000000;
// whitelisted by GMX, stablecoin, can be used for short positions:
  address tokenDAI = 0xDA10009cBd5D07dd0CeCc66161FC93D7c9000da1;

  pathShort = [address(tokenUSDC), tokenDAI];
  (address adapter, , ) = openPositions(amountIn, pearFee);

  uint256[] memory data = gmxFactory.getPosition(adapter);
  assertGt(data[0], 0);
  assertGt(data[9], 0);
```

```
    assertEq(uint256(gmxFactory.positionDetails(adapter, user)), 1);
    assertEq(
      tokenUSDC.balanceOf(treasury) + tokenUSDC.balanceOf(staking),
      pearFee
    );
    assertEq(tokenUSDC.balanceOf(staking), 0);
}
```

Location of Affected Code

File: src/Factory/GmxFactory#L172-L192

File: src/Factory/GmxFactory.sol#L635-L642

Recommendation

If the protocol is meant to not perform any short path swaps, it should prevent it. Otherwise, the collateral token of Pear positions should be set to the first element of the short path, not the last one.

Generally, any possibility of transaction revert during GMX callback will have the same consequences. Hence, this is crucial to verify all possible paths of reverts. Generally, pull payments should be preferred to push payments, even considering `ERC20`, which in the case of `USDC`, `USDT` and many others are upgradeable and may result in unexpected reverts in the future.

Team Response

Acknowledged and fixed as suggested.

[H-04] The `withdrawClosedSize()` Function Misses the Situation Where the Funds Are Transferred as `ETH` From the `GMX`

Severity

High Risk

Description

Users can opt to close their positions with ETH being received, accordingly, they can pass the flag:

File: src/interfaces/IPearGmxFactory.sol#L48-L49

```
struct DecreasePositionsArgs {
// @audit Flag to pass true
  bool withdrawETHLong;
// @audit Flag to pass true
  bool withdrawETHShort;
  ...
}
```

The same call is interpreted on the GMX side as `request.withdrawETH = true`.

Once the position is decreased and closed at the GMX side, the below function is called:

```
function executeDecreasePosition(bytes32 _key, address payable
    _executionFeeReceiver) public nonReentrant returns (bool) {
// code
  if (amountOut > 0) {
    if (request.path.length > 1) {
      IERC20(request.path[0]).safeTransfer(vault, amountOut);
      amountOut = _swap(request.path, request.minOut, address(this));
    }

    if (request.withdrawETH) { // @audit The intention is handled here;
      _transferOutETHWithGasLimitFallbackToWeth(amountOut, payable(
          request.receiver));
    } else {
      IERC20(request.path[request.path.length - 1]).safeTransfer(request.
          receiver, amountOut);
    }
  }
// code
}
```

At the `GmxFactory.sol` side, the closing position is handled during the callback as below:

File: src/Factory/GmxFactory.sol#L646-L667

```
} else if (
  longExecutionState == ExecutionState.Success ||
  shortExecutionState == ExecutionState.Success
) {
  if (!isIncrease) {
    uint256[] memory data = getPosition(adapter);
    if (
      positionDetails[adapter][adapterOwner] ==
      PositionStatus.Opened &&
      data[1] == 0 &&
      data[10] == 0
    ) {
      positionDetails[adapter][adapterOwner] = PositionStatus
        .Closed;
    }

    IPearGmxAdapter(adapter).withdrawClosedSize(
      feeAmount,
      platformLogic,
      collateralToken
    );
  }
// code
}
```

The `withdrawClosedsize()` function is below:

File: src/GmxAdapter.sol#L161

```
function withdrawClosedSize(
  uint256 feeAmount,
  address platformLogic,
  address token
) external onlyFactory {
  if (IERC20(token).balanceOf(address(this)) > 0) {
    SafeTransferLib.safeTransfer(token, platformLogic, feeAmount);

    SafeTransferLib.safeTransferAll(token, owner);
  }
}
```

So if the `withdrawETH()` flag is `true`, the `` `withdrawClosedSize() `` implementation doesn't cover this. Since it's not pushed to the user and to the Pear Side the `adapterOwner` can call below and get away with the platform fees.

File: src/GmxAdapter.sol#L186

```
/// @inheritdoc IPearGmxAdapter
function withdrawETH(
  address to,
  uint256 amount
) external override onlyOwner returns (bool success) {
  SafeTransferLib.safeTransferETH(to, amount);

  emit EthWithdrawal(to, amount);
  return true;
}
```

Impact

Loss of protocol fees.

Location of Affected Code

File: src/GmxAdapter.sol#L161-L171

```
function withdrawClosedSize(
  uint256 feeAmount,
  address platformLogic,
  address token
) external onlyFactory {
  if (IERC20(token).balanceOf(address(this)) > 0) {
    SafeTransferLib.safeTransfer(token, platformLogic, feeAmount);

    SafeTransferLib.safeTransferAll(token, owner);
  }
}
```

Recommendation

It's recommended to cover the situation where the funds are requested in native tokens.

Team Response

Acknowledged and fixed by removing withdrawal ETH functionality.

[H-05] Vertex Fees Transferred From User Do Not Account for Referee Fees, Leaving The Protocol with Permanent Shortfall

Severity

High Risk

Description

The vulnerability exists in the Vertex fee transfer process of the GMX protocol. In `GMX`, fees are first transferred to the `PlatformLogic.sol` contract, and then split among treasury and stakers, accounting for referee fees. However, in Vertex, fees intended for referrers are calculated and accounted for via the `PlatformLogic.addTokenFeesForWithdrawal()` function, but are not actually deducted from the amount transferred from users to stakers and treasury. This results in a deficit when referrers attempt to retrieve their fees, as the protocol lacks sufficient assets to fulfil these claims. Additionally, if the `PlatformLogic.sol` contract holds intermediate balances of pending operations, and a referrer retrieves their funds, the state update on these pending operations will fail, causing further issues.

Impact

Financial losses for the protocol due to the inability to pay referee fees from the protocol's funds.

Location of Affected Code

File: src/PlatformLogic.sol#L728

```
function _applyPlatformFeeErc20Vertex(
  address _referee,
  uint256 _grossAmount,
  IERC20 _tokenAddress
) internal {
  if (comptroller.getPlatformLogic() != address(this)) {
    revert PlatformLogic_AddressSetInComptrollerIsNotThisOne();
  }


  uint256 _feeAmount = calculateFees(_grossAmount, platformFee);

  if (referredUsers[_referee] != 0) {
    uint256 _refereeDiscount = calculateFees(
      _feeAmount,
      refereeDiscount
    );
```

```
    uint256 _referrerWithdrawal = calculateFees(
      _feeAmount,
      referrerFee
    );

    _feeAmount -= _refereeDiscount;

// code
    addTokenFeesForWithdrawal(
      checkReferredUser(_referee),
      _referrerWithdrawal,
      _tokenAddress
    );

    _feeAmount -= _referrerWithdrawal;

    uint256 _amountToBeSendToTreasury = calculateFees(
      _feeAmount,
      treasuryFeeSplit
    );

    SafeTransferLib.safeTransferFrom(
      address(_tokenAddress),
      _referee,
      PearTreasury,
      _amountToBeSendToTreasury
    );
  }
// code
}
```

Recommendation

Account for referee fees when taking fees for Vertex.

Team Response

Acknowledged and fixed as suggested.

[M-01] Current Fee Structure May Harm the Users and Allows for Fee Exploitation

Severity

Medium Risk

Description

The vulnerability lies in the way fees are calculated in the `GmxFactory.sol` and `NftHandler.sol` contracts of the Pear protocol. The fee calculation uses position size, which is a leverage. In the case

of for example **10x** leverage, users are supposed to pay **10x** more fees with the same collateral. Additionally, in most cases, the fees are either too big or too little. It stems from the fact that the fees are calculated based on current position size during `increase/decrease/close` operations.

```
/// @inheritdoc IPearGmxFactory
function createIncreasePositions(IncreasePositionsArgs memory args)
    external payable override returns (
  address adapter,
  bytes32 longPositionId,
  bytes32 shortPositionId
) {
// code
uint256[] memory data = getPosition(args.adapter);

uint256 amount = args.amountInLong + args.amountInShort;

/// @dev X * 1e6 / 1e30 = 1e24 - this will save some gas in calculation
uint256 sizeAmount = (data[0] + data[9]) / 1e24; // @audit it concerns
    the current position in GMX, not the one that will be created in this
    transaction

uint256 feeAmount = IPlatformLogic(platformLogic)
  .applyPlatformFeeErc20GmxManagePosition(
    msg.sender,
    sizeAmount,
    IERC20(collateralToken),
    address(this)
  );
// code
}
```

This allows for 2 abuses:

1. If the user has big leverage and adds just a small amount of collateral, he'll have to pay a huge fee calculated based on his current leveraged position value, possibly bigger than the position.
2. The user can exploit it and first create a very small position, paying an insignificant fee, then increase it with huge capital, paying for the previous size amount.

Impact

This vulnerability can have multiple impacts:

1. **Disproportionate Fees for Small Increases**: When increasing positions, users might pay disproportionately high fees compared to their position size, especially if they add a small amount to a large leveraged position.
2. **Exploitation of Fee Structure**: Users might first create a small position to pay a minimal fee and then significantly increase their position, effectively bypassing the intended fee structure.

Attack Scenario

Attackers or regular users could exploit this vulnerability in several ways:

1. **Fee Evasion**: By sending small amounts that in the fee calculation, users can perform actions without paying the intended fees.

2. **Exploiting Fee Discrepancies**: Users could create small initial positions with minimal fees and then add substantial capital, avoiding higher fees.
3. **Manipulating Frontend Calculations**: Users might face transaction reverts due to mismatches in fee calculations between the front end and the smart contract.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L217

File: src/Factory/GmxFactory.sol#L314

File: src/Factory/GmxFactory.sol#L409

File: src/Factory/GmxFactory.sol#L485

Recommendation

1. Don't use stale GMX position data for calculating the amount of fees.
2. Consider taking fees based on the amount increased/decreased in `increasePosition` and `decreasePosition` .
3. Reconsider if the user's leverage should correlate 1:1 with fees taken by the protocol. High fees will disincentivize users from using the protocol.

Team Response

Acknowledged and fixed as suggested.

[M-02] Transferring the Position NFT While There Is an Unexecuted Position Might Lose the Fees of The New Adapter Owner

Severity

Medium Risk

Description

The `updateOwner()` function fails to sync all the owner-related mappings.

Let's say a user has already a position and calls the `createIncreasePositions()` function: Meanwhile, the user sells their position NFT and the NFT is transferred to the new admin who supposedly owns the existing position and the callback which is to be received from the GMX side.

Let's assume that the execution has failed on the GMX side which is very easy due to various requirements from the GMX side and the fallback triggers to refund the funds; it will trigger as below:

File: src/Factory/GmxFactory.sol#L527-L549

```
if (
// main 1
  longExecutionState > ExecutionState.Pending &&
  shortExecutionState > ExecutionState.Pending
) {
// If both long/short fails either on increase or decrease
  if (
    longExecutionState == ExecutionState.Failed &&
    (shortExecutionState == ExecutionState.Failed)
  ) {
    IPearGmxAdapter(adapter).closeFailedPosition(
      pairData.short.path,
      adapterOwner
    );
    _handleFeeAmount(
      adapterOwner,
      adapter,
      platformLogic,
      collateralToken,
      feeAmount,
      true
    );
  } else if (
}
```

And `_handleFeeAmount()` function will be called as below:

File: src/Factory/GmxFactory.sol#L458

```
/// @dev Difference in fee logic based on failed or success flag. Being
    used in several places in the gmxPositionCallback
function _handleFeeAmount(
  address _referee,
  address adapter,
  address platformLogic,
  address collateralToken,
  uint256 feeAmount,
  bool isFailed
) internal {
  if (isFailed) { // @audit the call will follow this path
    IPlatformLogic(platformLogic)
      .handleTokenAmountWhenPositionHasFailed(
        _referee,
        adapter,
        feeAmount,
        IERC20(collateralToken)
      );
```

```
  } else {
    IPlatformLogic(platformLogic).splitBetweenStakersAndTreasuryToken(
      _referee,
      adapter,
      feeAmount,
      IERC20(collateralToken)
    );
  }
}
```

Since `isFailed = true`, `handleTokenAmountWhenPositionHasFailed()` function will be called:

File:src/PlatformLogic.sol#L314

```
// @inheritdoc IPlatformLogic
function handleTokenAmountWhenPositionHasFailed(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount,
  IERC20 _tokenAddress
) external override onlyFactory {
  if (refereeFeeAmounts[_referee][_adapterAddress] == 0) { // @audit
     _referee is the new adapterOwner
    revert PlatformLogic_FeeAmountCannotBeNull();
  }
  setRefereeFeeAmount(_referee, _adapterAddress, 0);

  SafeTransferLib.safeTransfer(
    address(_tokenAddress),
    _referee,
    _feeAmount
  );
}
```

The `gmxPositionCallback` takes the `adapterOwner` from these lines below and continues to use it in the if conditions as being the `_referee` input:

Contract: src/Factory/GmxFactory.sol#L502

```
address adapterOwner = adapterOwners[adapter];
```

Since the `_referee` is the new `adapterOwner`, L:293-294 will revert the transaction because the new `adapterOwner` didn't perform any call, and the new `adapterOwner` will not be receiving the fees for the failed position.

In addition:

Minting the position to `address _to` (the new `adapterOwner`) while having a pending execution state will create the same issue. So the user might also utilize the option of minting the position NFT to their other address and failing to receive the fee.

Impact

Loss of funds after NFT transfer.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L694-#L711

```
/// @inheritdoc IPearGmxFactory
function updateOwner(
  address oldOwner,
  address newOwner,
  address adapter
) external override returns (bool) {
  address nftHandler = comptroller.getNftHandler();
  if (msg.sender != nftHandler) {
    revert Errors.GmxFactory_NotNftHandler();
  }
  adapterOwners[adapter] = newOwner;
  positions[newOwner] += 1;
  positions[oldOwner] -= 1;
  indexedPositions[newOwner][positions[newOwner]] = adapter;
  positionDetails[adapter][oldOwner] = PositionStatus.Transferred;
  positionDetails[adapter][newOwner] = PositionStatus.Opened;

  return true;
}
```

Recommendation

It's recommended to sync the `refereeFeeAmounts[_referee][_adapterAddress]` as well in the `updateOwner()` call.

Team Response

Acknowledged and fixed as suggested.

[M-03] The Referrer Receives the Fees Even If the Position Execution Fails

Severity

Medium Risk

Description

If a user is accepting a referrer to have fee discounts, it might not be the best for them due to some conditions. The referrer fee is taken every time from the referee whenever the referee has a position interaction.

File: src/PlatformLogic.sol

```
// code

// calculate the referrer discount
// for testing the referralFee is set to 5 bps
uint256 _referrerWithdrawal = calculateFees(
  _feeAmount,
  referrerFee
);

//The fees are written to the referrer's pendingTokenWithdrawals mapping
addTokenFeesForWithdrawal(
  checkReferredUser(_referee),
  _referrerWithdrawal,
  _tokenAddress
);
// code
```

File: src/PlatformLogic.sol#L305-L314

```
/// @inheritdoc IPlatformLogic
function addTokenFeesForWithdrawal(
  address _referrer,
  uint256 _amount,
  IERC20 _token
) public override notZeroAddress(_referrer) onlyFactory {
// add a withdrawTokenFee function
  pendingTokenWithdrawals[_referrer][_token] += _amount;
  emit PendingTokenWithdrawal(_referrer, _amount);
}
```

The final discounted fee is registered as below:

File: src/Factory/GmxFactory.sol#L143

```
IPlatformLogic(platformLogic).setRefereeFeeAmount(
  msg.sender, //  <-- referee
  adapter,
  feeAmount // <-- discounted fee
);
```

However, if the position fails to be executed at the GMX side, only the registered fee is returned to the user.

File: src/Factory/GmxFactory.sol#L450

```
/// @dev Difference in fee logic based on failed or success flag. Being
    used in several places in the gmxPositionCallback
function _handleFeeAmount(
  address _referee,
  address adapter,
  address platformLogic,
  address collateralToken,
  uint256 feeAmount,
  bool isFailed
) internal {
  if (isFailed) {
    IPlatformLogic(platformLogic)
      .handleTokenAmountWhenPositionHasFailed(
        _referee,
        adapter,
        feeAmount,
        IERC20(collateralToken)
      );
  } else {
    IPlatformLogic(platformLogic).splitBetweenStakersAndTreasuryToken(
      _referee,
      adapter,
      feeAmount,
      IERC20(collateralToken)
    );
  }
}
```

File: src/PlatformLogic.sol#L287

```
/// @inheritdoc IPlatformLogic
function handleTokenAmountWhenPositionHasFailed(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount,
  IERC20 _tokenAddress
) external override onlyFactory {
  if (refereeFeeAmounts[_referee][_adapterAddress] == 0) {
    revert PlatformLogic_FeeAmountCannotBeNull();
  }
    setPendingReferrerFeeAmount( _adapterAddress, checkReferredUser(
       _referee),0);
    setRefereeFeeAmount(_referee, _adapterAddress, 0);
  SafeTransferLib.safeTransfer(address(_tokenAddress),_referee,_feeAmount
     );
}
```

While Pear Protocol doesn't incur fees in this situation, the referrer takes the fees even if the position fails. Not incurring fees can be a design choice for the Pear Protocol, however, through the user experience and trading logic side, failed orders don't incur fees.

Impact

Users will lose funds for the positions they don't possess.

Location of Affected Code

File: src/PlatformLogic.sol#L287-303

```
function handleTokenAmountWhenPositionHasFailed(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount,
  IERC20 _tokenAddress
) external override onlyFactory {
  if (refereeFeeAmounts[_referee][_adapterAddress] == 0) {
    revert PlatformLogic_FeeAmountCannotBeNull();
  }
  setRefereeFeeAmount(_referee, _adapterAddress, 0);

  SafeTransferLib.safeTransfer(
    address(_tokenAddress),
    _referee,
    _feeAmount
  );
}
```

Recommendation

The referrer fees could be updated during a successful callback rather than performing it at the beginning.

Team Response

Acknowledged and fixed as suggested.

[M-04] The Protocol Fails to Create a Unique Salt in Some Conditions

Severity

Medium Risk

Description

The `GmxFactory.openPositions()` function uses salt to create the adapter as below:

```
/// @inheritdoc IPearGmxFactory
function openPositions(OpenPositionsArgs memory args) external payable
    override returns (
  address adapter,
  bytes32 longPositionId,
  bytes32 shortPositionId
) {
// code
uint256 current = positions[msg.sender];
bytes32 salt = keccak256(
  abi.encodePacked(
    msg.sender,
    args.indexTokenLong,
    args.indexTokenShort,
    current + 1
  )
);
// code
}
```

Line:117: `current + 1`, uses the current value that holds the user's position count.

However `GmxFactory.updateOwner()` updates it in the opposite direction:

```
/// @inheritdoc IPearGmxFactory
function updateOwner(
  address oldOwner,
  address newOwner,
  address adapter
) external override returns (bool) {
  address nftHandler = comptroller.getNftHandler();
  if (msg.sender != nftHandler) {
    revert Errors.GmxFactory_NotNftHandler();
  }
  adapterOwners[adapter] = newOwner;
  positions[newOwner] += 1;
  positions[oldOwner] -= 1; // @audit Decrements the position count
  indexedPositions[newOwner][positions[newOwner]] = adapter;
  positionDetails[adapter][oldOwner] = PositionStatus.Transferred;
  positionDetails[adapter][newOwner] = PositionStatus.Opened;

  return true;
}
```

Line: 705: `positions[oldOwner] -= 1`, decrements the position count.

So the next position with the same `args.indexTokenLong`, `args.indexTokenShort` will revert due to `current + 1` will be the same since there's an adapter already with the same address.

Impact

The user can't open a position with the same tokens for the next trade.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L705

```
/// @inheritdoc IPearGmxFactory
function updateOwner(
  address oldOwner,
  address newOwner,
  address adapter
) external override returns (bool) {
// code
  positions[oldOwner] -= 1;
// code
}
```

Recommendation

A better approach could be based on a new variable that holds the user's number of `openPositions()` function calls count rather than the existing positions.

Team Response

Acknowledged and fixed as suggested.

[M-05] NFT Transfer Disallowed When Any of the Side Is Changed to Zero

Severity

Medium Risk

Description

The vulnerability exists in the `transferNft()` function, which is designed to transfer a position represented as an NFT from one user to another. The issue arises due to the condition that checks if either the long or short position (`data[0]` or `data[9]`) is 0, leading to a revert with the `NftHandler_PositionNonExistantOrAlreadyClosed` error. This check is flawed because there are legitimate scenarios where one leg of a position (either long or short) can be 0 without the position being non-existent or closed. These scenarios include:

- A user intentionally decreasing one leg of their position to 0 to mitigate loss risk.
- One leg of the position is being liquidated by GMX.

Additionally, this vulnerability can be exploited maliciously. An attacker could decrease their position on either long or short to 0 and then transfer the NFT in an NFT lending protocol. Since the position still reflects the old values until the GMX order is executed, the NFT can be incorrectly transferred, leading to issues when the protocol tries to lend out the NFT to other users.

Impact

The impact of this vulnerability includes:

- **Legitimate Transactions Blocked**: Users cannot transfer positions where one leg is legitimate 0, limiting their ability to manage their investments.
- **Potential for Exploitation**: Malicious users can exploit this flaw to transfer NFTs that should be otherwise non-transferable, potentially causing disruptions in NFT lending protocols.
- **Incorrect Representation of Position Status**: The current check may inaccurately represent the status of a position, leading to confusion and erroneous transaction reverts.

Location of Affected Code

File: src/NFT/NftHandler.sol#L163-L187

```
/// @inheritdoc INFTHandler
function transferNft(uint256 tokenId, address to) external override {
  address gmxFactory = comptroller.getGmxFactory();
  address positionNft = comptroller.getPositionNft();

  if (IPositionNFT(positionNft).ownerOf(tokenId) != msg.sender) {
    revert NftHandler_NotNftOwner();
  }

  address adapter = tokenIds[tokenId];

  uint256[] memory data = IPearGmxFactory(gmxFactory).getPosition(
    adapter
  );

  if (data[0] == 0 || data[9] == 0) {
    revert NftHandler_PositionNonExistantOrAlreadyClosed();
  }

// Transfer the NFT using the PositionNFT contract
  IPositionNFT(positionNft).safeTransferFrom(msg.sender, to, tokenId);
  IPearGmxAdapter(adapter).changePositionOwner(to);
  IPearGmxFactory(gmxFactory).updateOwner(msg.sender, to, adapter);

  emit UpdateOwner(msg.sender, to, adapter);
}
```

Recommendation

Switch the condition from `data[0] == 0 || data[9] == 0` to `data[0] == 0 && data[9] == 0`, or allow users to transfer the even closed NFTs.

Team Response

Acknowledged and fixed as suggested.

[M-06] Fees Sent to `PlatformLogic.sol` Contract While Closing Position Do Not Account For Insufficient Adapter Funds

Severity

Medium Risk

Description

In case a position is closed, function `withdrawClosedSize()` is called to send fees from the `adapter` to the `PlatformLogic.sol` contract, in order to pay fees with it:

The problem is that the position may not have enough funds to send to the `PlatformLogic.sol`. This results in transaction reverting and the position state not being correctly handled.

Impact

GMX callback revert resulting in position state not being correctly handled. The leftover funds that would be available for partial repayment are not taken by the Pear protocol and are left for the user.

Location of Affected Code

File: src/GmxAdapter.sol#L174

```
  /// @inheritdoc IPearGmxAdapter
function withdrawClosedSize(
  uint256 feeAmount,
  address platformLogic,
  address token
) external onlyFactory {
  if (IERC20(token).balanceOf(address(this)) > 0) {
    SafeTransferLib.safeTransfer(token, platformLogic, feeAmount); //
       @audit Adapter may not have enough funds for paying the fees

    SafeTransferLib.safeTransferAll(token, owner);
  }
}
```

Recommendation

Update the `withdrawClosedSize()` function to include a check ensuring that the `GmxAdapter` has enough funds to cover the fees before attempting the transfer. In case of insufficient funds, establish a clear and transparent fallback mechanism, like taking whatever is in the contract if it's not enough to cover fees.

Team Response

Acknowledged and fixed as suggested.

[M-07] Reusing the Same `minOut` For Long and Short Position Makes Slippage Protection Ineffective, and May Result in DoS for Opening New Positions

Severity

Medium Risk

Description

When opening a position, the initial token for long and short positions has to be the same. Pear uses swap paths in order to comply with requirements from GMX:

If it's a long position, then:

- collateral has to be an index token
- collateral can't be stablecoin

Otherwise:

- collateral has to be stablecoin
- collateral can't be an index token
- index token has to be shortable

In case of a swap, GMX checks if the `minOut` is as expected after the swap in `BasePositionManager`:

```
function _vaultSwap(address _tokenIn, address _tokenOut, uint256 _minOut,
    address _receiver) internal returns (uint256) {
  uint256 amountOut = IVault(vault).swap(_tokenIn, _tokenOut, _receiver);
  require(amountOut >= _minOut, "insufficient amountOut");
  return amountOut;
}
```

It works in the test cases because the tests don't account for the fact that the short path may have a swap path, and GMX skipped `minOut` in this case.

However, Pear allows for a swap path on both long and short tokens and reuses the same `minOut` for both GMX positions.

This may lead to 2 possible consequences, in case of large long and short path token discrepancies, let's say 6 decimals and 18 decimals:

- If `minOut` of 6 decimals token is used, then 18 decimals token slippage protection literally doesn't exist, as it would have to drop to pennies to be effective
- If `minOut` of 18 decimals token is used, then the position will be reverted during execution because it will never achieve a given amount

Impact

This vulnerability undermines the effectiveness of slippage protection in the Pear protocol, potentially leading to substantial financial losses for users due to inadequate slippage control. Moreover, it could result in the consistent failure of new position openings, particularly in scenarios with significant discrepancies between the decimal places of long and short path tokens. This could lead to a Denial of Service situation, where users are unable to execute their trading strategies effectively.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L159-L178

```
function openPositions(OpenPositionsArgs memory args) external payable
    override returns (
  address adapter,
  bytes32 longPositionId,
  bytes32 shortPositionId
) {
  if (args.pathLong[0] != args.pathShort[0]) {
    revert Errors.GmxFactory_DifferentCollateralToken();
  }
  if (args.indexTokenLong == args.indexTokenShort) {
    revert Errors.GmxFactory_SameIndexToken();
  }
  if (comptroller.allowedTokens(IERC20(args.pathLong[0])) != true) {
    revert Errors.GmxFactory_TokenNotAllowed();
  }

// code
  longPositionId = openPosition(
    adapter,
    args.pathLong,
    args.indexTokenLong,
    args.amountInLong,
    args.minOut, // @audit minOut is reused for short and long token
    true,
    args.sizeDeltaLong,
    args.acceptablePriceLong
  );
  shortPositionId = openPosition(
    adapter,
    args.pathShort,
    args.indexTokenShort,
    args.amountInShort,
    args.minOut, // @audit minOut is reused for short and long token
    false,
    args.sizeDeltaShort,
    args.acceptablePriceShort
  );
// code
}
```

Recommendation

Implement distinct `minOut` calculations for long and short positions, taking into account the decimal differences of the respective tokens. However, as described in another finding - **_Insufficient Short Path Validation Locks Fees in PlatfromLogic.sol And Prevents Any State Change of a Position_**- short position swap path currently leads to locking user funds, hence we recommend reconsidering the usage of short swap path altogether.

Team Response

Acknowledged and fixed as suggested.

[M-08] No Slippage Protection for `Decrease/Close` Position May Create a Trade That is Unfavourable to The User

Severity

Medium Risk

Description

Opening position and increasing it has slippage protection while decreasing it and closing doesn't, which may execute the trade at a price unfavourable to the user.

Impact

Funds loss due to unfavourable trade for the user.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L908

```
function createDecreasePosition(
  address adapter,
  address indexToken,
  address[] memory path,
  uint256 amount,
  uint256 acceptablePrice,
  bool isLong,
  bool _withdrawETH
) internal returns (bytes32 positionId) {
// code
  positionId = IPearGmxAdapter(adapter).createDecreasePosition{
    value: executionFee
  }(
    DecreasePositionArgs({
      path: path,
      amountIn: amount,
      indexToken: indexToken,
      receiver: msg.sender,
      sizeDelta: 0,
      minOut: 0, // @audit no slippage protection
      withdrawETH: _withdrawETH,
      acceptablePrice: acceptablePrice,
      isLong: isLong,
      callbackTarget: address(this),
      executionFee: executionFee
    })
  );
// code
}
```

File: src/Factory/GmxFactory.sol#L966

```
function closePosition(
  address adapter,
  address[] memory path,
  uint256 acceptablePrice,
  address indexToken,
  uint256 sizeDelta,
  uint256 amountIn,
  bool isLong,
  bool _withdrawETH,
  uint256[] memory data
) internal returns (bytes32 positionId) {
// code

/// @dev runs if position was executed
  if (size > 0) {
    DecreasePositionArgs memory args = DecreasePositionArgs({
      path: path,
      amountIn: amountIn,
      indexToken: indexToken,
      sizeDelta: sizeDelta,
      receiver: adapter,
      acceptablePrice: acceptablePrice,
      withdrawETH: _withdrawETH,
      isLong: isLong,
      minOut: 0, // @audit no slippage protection
      callbackTarget: address(this),
      executionFee: executionFee
    });
  }
// code
}
```

Recommendation

Add slippage protection the same way as for increased position and open position operations.

Team Response

Acknowledged and fixed as suggested.

[M-09] `PositionNft` Cannot Be Transferred By Itself

Severity

Medium Risk

Description

Pear docs say that the position NFTs are composable with other systems, however, position transfer logic is implemented in `NFTHandler.sol`, which makes it not composable. NFT can-

not be sent directly via `PositionNFT.transferFrom/safeTransferFrom`, because the original `_isApprovedOrOwner()` is overridden. The original:

```
function _isApprovedOrOwner(address spender, uint256 tokenId) internal
    view virtual returns (bool) {
  address owner = ownerOf(tokenId);
  return (spender == owner || isApprovedForAll(owner, spender) ||
      getApproved(tokenId) == spender);
}
```

Pear version:

```
function _isApprovedOrOwner(
  address,
  uint256
) internal view virtual override onlyNftHandler returns (bool) {
  return true; // @audit Only Nft Handler is allowed, no approvals will
      work
}
```

So, the core mechanic of approvals doesn't work with Pear NFT.

Impact

NFT, contrary to the docs, is not composable with DeFi and cannot be used in other protocols, because it lacks the means to approve external smart contracts to transfer NFTs.

Location of Affected Code

File: src/NFT/NftHandler.sol#L163-L187

```
/// @inheritdoc INFTHandler
function transferNft(uint256 tokenId, address to) external override {
  address gmxFactory = comptroller.getGmxFactory();
  address positionNft = comptroller.getPositionNft();

  if (IPositionNFT(positionNft).ownerOf(tokenId) != msg.sender) {
    revert NftHandler_NotNftOwner();
  }

  address adapter = tokenIds[tokenId];

  uint256[] memory data = IPearGmxFactory(gmxFactory).getPosition(
    adapter
  );
```

```
    if (data[0] == 0 || data[9] == 0) {
      revert NftHandler_PositionNonExistantOrAlreadyClosed();
    }

// Transfer the NFT using the PositionNFT contract
    IPositionNFT(positionNft).safeTransferFrom(msg.sender, to, tokenId);
    IPearGmxAdapter(adapter).changePositionOwner(to);
    IPearGmxFactory(gmxFactory).updateOwner(msg.sender, to, adapter);

    emit UpdateOwner(msg.sender, to, adapter);
}
```

Recommendation

There are two possible cases where it can be handled:

1. Move position ownership transfer logic from `NftHandler.sol` to `PositionNFT.sol`.
2. Add callback function in `NftHandler.sol`, which changes position ownership when any transfer of `PositionNFT.sol` is performed.

Additionally, approvals are not working with the current implementation, `NftHandler.sol` should respect that.

Team Response

Acknowledged and fixed as suggested.

[M-10] The Old Position Owner Could Cause Damage to The New Position Owner

Severity

Medium Risk

Description

The position of NFT could be utilized in fund transfers as stated in the protocol docs. A user holding a good position might want to cash out by selling their NFT without closing the position.

However, the new owner might be in a position where the old owner has already a pending position to be executed at the GMX side either for increasing or decreasing the position. While the new owner candidate can't know the status, the old owner could grief the new owner either with;

- Leveraging the position close to being liquidated - so that the owner would harm the new user by spending some funds to perform this action
- Bringing the position to dust amounts so that they can have the possibility to batch a TX with `(withdrawToken(all) + transferNFT)` after the position is decreased - Subject to extremely accurate timing to back run GMX fallback call in the same block. But since there's a `decreasePosition` order on the GMX side, the new position owner will not benefit from buying the position NFT anyways.

Impact

Loss of funds.

Location of Affected Code

File: src/NFT/NftHandler.sol#L163-L187

```
/// @inheritdoc INFTHandler
function transferNft(uint256 tokenId, address to) external override {
  address gmxFactory = comptroller.getGmxFactory();
  address positionNft = comptroller.getPositionNft();

  if (IPositionNFT(positionNft).ownerOf(tokenId) != msg.sender) {
    revert NftHandler_NotNftOwner();
  }

  address adapter = tokenIds[tokenId];

  uint256[] memory data = IPearGmxFactory(gmxFactory).getPosition(
    adapter
  );

  if (data[0] == 0 || data[9] == 0) {
    revert NftHandler_PositionNonExistantOrAlreadyClosed();
  }

// Transfer the NFT using the PositionNFT contract
  IPositionNFT(positionNft).safeTransferFrom(msg.sender, to, tokenId);
  IPearGmxAdapter(adapter).changePositionOwner(to);
  IPearGmxFactory(gmxFactory).updateOwner(msg.sender, to, adapter);

  emit UpdateOwner(msg.sender, to, adapter);
}
```

Recommendation

We recommend disabling NFT transfers if the user has a pending position on the GMX side.

Team Response

Acknowledged and fixed as suggested.

[M-11] If `NFT` Is Transferred to Current Owner, It Is Lost

Severity

Medium Risk

Description

The vulnerability arises in the `updateOwner()` function, designed to transfer ownership of an adapter between two addresses within a system managing positions through NFTs. The function incorrectly handles the case where the `oldOwner` and `newOwner` are the same. Specifically, after updating the `adapterOwners` mapping to reflect the new owner and incrementing the `positions` count for the `newOwne` , it attempts to zero out the position for the `oldOwner` . However, when `oldOwner` and `newOwner` are the same, this action inadvertently zeroes out the newly updated position, leading to the loss of the adapter's tracking and ownership details within the `indexedPositions` mapping.

Impact

Loss of ownership and funds assigned to a position.

Attack Scenario

1. `oldOwner` initiates `updateOwner(oldOwner, oldOwner, adapter)` .
2. The system correctly identifies `msg.sende` ' as the position NFT controller and proceeds.
3. `adapterOwners[adapter]` is set to `oldOwner` (no change).
4. `positions[oldOwner]` is incremented.
5. `indexedPositions[oldOwner][positions[oldOwner]]` is set to `address(0)`, zeroing out the newly incremented position due to the flawed logic.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L819

```
function updateOwner(
  address oldOwner,
  address newOwner,
  address adapter
) external override returns (bool) {
  address positionNft = comptroller.getPositionNft();
  if (msg.sender != positionNft) {
    revert Errors.GmxFactory_NotPositionNFT();
  }
  adapterOwners[adapter] = newOwner;
  positions[newOwner] += 1;
  indexedPositions[newOwner][positions[newOwner]] = adapter;
  indexedPositions[oldOwner][positions[oldOwner]] = address(0); // @audit
      If oldOwner == newOwner, position will be zeroed out
  positionDetails[adapter][oldOwner] = PositionStatus.Transferred;
  positionDetails[adapter][newOwner] = PositionStatus.Opened;

  return true;
}
```

Recommendation

Either don't allow the old owner to be the new owner, or swap line position of `indexedPositions[newOwner][positions[newOwner]] = adapter;` with `indexedPositions[oldOwner][positions[oldOwner]] = address(0);`.

Team Response

Acknowledged and fixed as suggested.

[M-12] Introduction of Global Referrer Fees Leads to Unexpected Reverts And Inability to Create Certain Positions

Severity

Medium Risk

Description

The vulnerability is located in the `openPositions()` function, where the introduction of global referrer fees leads to incorrect fee calculations and unexpected behaviors. The function calculates `referrerAmount` based on pending referrer fee amounts and adjusts the `args.amountInLong` and `args.amountInShort` by subtracting the `referrerAmount`. However, the function applies the total fee calculated from the sum of long and short positions to each part individually, effectively double-charging fees. This mechanism also fails in scenarios involving one-sided position modifications, where either a long or short position is adjusted independently.

Impact

This issue can lead to unexpected reverts, preventing users from creating certain positions.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L174-L188

```
/// @inheritdoc IPearGmxFactory
function openPositions(OpenPositionsArgs memory args) external payable
    override returns (
  address adapter,
  bytes32 longPositionId,
  bytes32 shortPositionId
) {
// code
```

```
  uint256 referrerAmount = IPlatformLogic(platformLogic)
    .pendingReferrerFeeAmounts(
      adapter,
      IPlatformLogic(platformLogic).checkReferredUser(msg.sender)
    );

  if (args.amountIn != amount + feeAmount) {
    revert Errors.GmxFactory_IncorrectGrossFeeAmount();
  }

  if (referrerAmount > 0) {
    args.amountInLong -= referrerAmount;
    args.amountInShort -= referrerAmount;
    amount = args.amountInLong + args.amountInShort;
  }
// code
}
```

Recommendation

The `openPositions()` function should be revised to individually calculate and apply fees for long and short positions, taking into account their respective values and denominations. This could involve separate fee calculations for each position type or a more dynamic fee distribution mechanism that accurately reflects the proportions of long and short positions.

Team Response

Acknowledged and fixed as suggested.

[M-13] Division Before Multiplication at GmxFactory's `feeSplit` Calculation

Severity

Medium Risk

Description

GmxFactory's `openPositions()` & `closePositions()` functions calculate the `feeSplit` as:

```
// The fees are expressed as a fraction of 10000, representing 100%.
feeSplit =
  (((args.sizeDeltaLong * 1e6) / 1e30) * 10000) /
  sizeAmount;
```

Since the calculations are made on division before multiplication fashion, this might end in the `feeSplit` being fewer than the normal maths due to truncation.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L154-#L156
File: src/Factory/GmxFactory.sol#L504-#L506

Recommendation

Recommend performing division at the end of the calculation.

Team Response

Acknowledged and fixed as suggested.

[L-01] Users Are Unable to Deleverage Their Positions

Severity

Low Risk

Description

In the `GmxFactory.createDecreasePosition()` function, the `sizeDelta` parameter is hardcoded to 0. This parameter is crucial for adjusting the leverage of a position. By being set to zero, it effectively prevents users from decreasing their leverage, which is a fundamental aspect of managing risk and avoiding liquidation, especially in volatile market conditions.

Impact

The inability to decrease leverage has several significant impacts:

1. **Increased Risk of Liquidation**: Users are unable to reduce their position size, which could be crucial in avoiding liquidation under adverse market conditions.
2. **Lack of Flexibility in Position Management**: This limitation restricts users' ability to dynamically manage their investments and adapt to changing market conditions.
3. **Potential Financial Losses**: Users might incur unnecessary financial losses due to the inability to decrease leverage in a declining market.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L907

```
function createDecreasePosition(
  address adapter,
  address indexToken,
  address[] memory path,
  uint256 amount,
  uint256 acceptablePrice,
  bool isLong,
  bool _withdrawETH
) internal returns (bytes32 positionId) {
// code
```

```
    positionId = IPearGmxAdapter(adapter).createDecreasePosition{
      value: executionFee
    }(
      DecreasePositionArgs({
        path: path,
        amountIn: amount,
        indexToken: indexToken,
        receiver: msg.sender,
        sizeDelta: 0, // @audit Hardcoded to zero, hence user not able to
           deleverage
        minOut: 0,
        withdrawETH: _withdrawETH,
        acceptablePrice: acceptablePrice,
        isLong: isLong,
        callbackTarget: address(this),
        executionFee: executionFee
      })
    );
  // code
  }
```

Recommendation

Consider modifying the `createDecreasePosition()` function to allow for a dynamic input for the `sizeDelta` parameter, rather than hardcoding it to 0.

Team Response

Acknowledged and fixed as suggested.

[L-02] Position State Can Be Broken If Referee Is Blacklisted

Severity

Low Risk

Description

There is an issue in the `PlatformLogic.handleTokenAmountWhenPositionHasFailed()` function, specifically related to the handling of USDC and USDT which have a blacklist feature. If a referee (recipient of funds) is blacklisted, any operations that attempt to send assets to their address will fail. This function is designed to handle the transfer of a specified `_feeAmount` of `_tokenAddress` (presumably USDC or USDT) to a referee's address. However, if the referee is on the blacklist of the token contract, the `SafeTransferLib.safeTransfer()` call will fail, leading to a breakdown in the asset management process within the contract.

Impact

Transactions will fail for blacklisted referees, leading to an inability to execute intended token transfers.

Location of Affected Code

File: src/PlatformLogic.sol#L287

```
/// @inheritdoc IPlatformLogic
function handleTokenAmountWhenPositionHasFailed(
  address _referee,
  address _adapterAddress,
  uint256 _feeAmount,
  IERC20 _tokenAddress
) external override onlyFactory {
  if (refereeFeeAmounts[_referee][_adapterAddress] == 0) {
    revert PlatformLogic_FeeAmountCannotBeNull();
  }
  setRefereeFeeAmount(_referee, _adapterAddress, 0);

  SafeTransferLib.safeTransfer(
    address(_tokenAddress),
    _referee, // @audit If referee is blacklisted, this will fail
    _feeAmount
  );
}
```

Recommendation

Replace the current push-based payment method with a pull payment system. In this system, instead of directly sending funds to users' addresses, the contract should allow users to withdraw their funds themselves.

Team Response

Acknowledged and fixed as suggested.

[L-03] The Next New Position Index of The Old Owner Could Overwrite The Existing Indexes of The Old Owner

Severity

Low Risk

Description

The `GmxFactory.updateOwner()` function syncs the mapping of the old owner to the new owner and adds a new position to the new owner's `indexedPositions` mapping. However, in some conditions where the old owner has multiple positions, the next new position index of the old owner could overwrite the existing indexes after the position NFT transfer.

Impact

Position Indexing logic is broken.

Attack Scenario

Let's say a user has 5 positions for different pairs, so the user is the owner of 5 adapters. `positions[user] = 5` at this point. The latest adapter is indexed as `indexedPositions[user][positions[user]] = adapter;` , so it's `indexedPositions[user][5]` .

1. The user sells his 4th NFT position and remains with 4 positions. Due to this, `GmxFactory.updateOwner()` decrements the position count `positions[user] -= 1` so it becomes 4.
2. The user opens another position and this time `positions[user]` becomes 5 and the user's position will be indexed as `indexedPositions[user][positions[user]] = adapter` . However, this is the index of the user's 5th position and it will be overwritten by this one.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L694-L711

```
function updateOwner(
  address oldOwner,
  address newOwner,
  address adapter
) external override returns (bool) {
  address nftHandler = comptroller.getNftHandler();
  if (msg.sender != nftHandler) {
    revert Errors.GmxFactory_NotNftHandler();
  }
  adapterOwners[adapter] = newOwner;
  positions[newOwner] += 1;
  positions[oldOwner] -= 1;
  indexedPositions[newOwner][positions[newOwner]] = adapter;
  positionDetails[adapter][oldOwner] = PositionStatus.Transferred;
  positionDetails[adapter][newOwner] = PositionStatus.Opened;

  return true;
}
```

Recommendation

While the indexing logic could be broken, we don't see this as a vulnerability with a heightened severity unless the code is implemented in a wider usage in the upcoming versions or a third party integrates to Pear Protocol.

We recommend keeping the indexes in an array and `positions` mapping should not be used in the `indexedPositions` mapping.

Team Response

Acknowledged and fixed as suggested.

[L-04] Missing Sanity Checks in `setTreasuryFeeSplit()` Function, Allows to Apply Over 100 Percent Fees

Severity

Low Risk

Description

The vulnerability lies in the `PlatformLogic.setTreasuryFeeSplit()` function of the smart contract, where there is a lack of sanity checks for the input value `_treasuryFeeSplit`. The function currently does not validate whether the input exceeds `10000` basis points (BPS), which represents `100 percent`. As a result, it is possible to set the treasury fee split to a value greater than 100%:

There are also other admin functions to set fees and fee splits don't have safeguards on how much they can charge, and technically they can go up to `100 percent` and beyond.

Impact

This will be very problematic when calling `PlatformLogic.splitBetweenStakersAndTreasuryToken()`, because exceeding this value will send current operations assets as well as pending operations' assets, resulting in other operations failing with ERC20 transfers during GMX callback, freezing its state.

Location of Affected Code

File: src/PlatformLogic.sol

```
/// @inheritdoc IPlatformLogic
function setTreasuryFeeSplit(
  uint256 _treasuryFeeSplit
) external override onlyAdmin {
  emit TreasuryFeeSplitChanged(treasuryFeeSplit, _treasuryFeeSplit);
  treasuryFeeSplit = _treasuryFeeSplit;
}
/// @inheritdoc IPlatformLogic
function setRefereeDiscount(
  uint256 _refereeDiscount
) external override onlyAdmin {
  emit RefereeDiscountChanged(refereeDiscount, _refereeDiscount);
  refereeDiscount = _refereeDiscount;
}

/// @inheritdoc IPlatformLogic
function setReferrerFee(uint256 _referrerFee) external override onlyAdmin
    {
  emit ReferrerFeeChanged(referrerFee, _referrerFee);
  referrerFee = _referrerFee;
}
```

```
/// @inheritdoc IPlatformLogic
function setPlatformFee(uint256 _platformFee) external override onlyAdmin
    {
  emit PlatformFeeChanged(platformFee, _platformFee);
  platformFee = _platformFee;
}

/// @inheritdoc IPlatformLogic
function setManagePositionFee(uint256 _fee) external override onlyAdmin {
  emit ManagePositionFeeChanged(managePositionFee, _fee);
  managePositionFee = _fee;
}

/// @inheritdoc IPlatformLogic
function setMintPositionFee(uint256 _fee) external override onlyAdmin {
  emit MintFeeChanged(mintFee, _fee);
  mintFee = _fee;
}


/// @inheritdoc IPlatformLogic
function setTreasuryFeeSplit(
  uint256 _treasuryFeeSplit
) external override onlyAdmin {
  emit TreasuryFeeSplitChanged(treasuryFeeSplit, _treasuryFeeSplit);
  treasuryFeeSplit = _treasuryFeeSplit;
}
```

Recommendation

Modify the `setTreasuryFeeSplit()` function to include a check that ensures `_treasuryFeeSplit` is not greater than 10000 BPS.

Team Response

Acknowledged and fixed as suggested.

[L-05] Admin Can Make a `Position NFT` Unmineable by `Blacklisting` Position Collateral Token

Severity

Low Risk

Description

When an NFT is minted, the function `NftHandler.mintNFT()` checks if the position short collateral token is in `Comptroller.allowedTokens()`:

```
/// @inheritdoc INFTHandler
function mintNFT(
  address adapter,
  address to
) external override returns (uint256 tokenId) {
// code

  PairPositionData memory pairData = IPearGmxFactory(gmxFactory)
    .getPairPositionData(adapter);
  address collateralToken = pairData.short.collateralToken;
  IERC20 _token = IERC20(collateralToken);

  if (comptroller.allowedTokens(_token) != true) { // @audit check if it'
     s allowed
    revert NFTHandler_TokenNotAllowed();
  }

// code
}
```

However, the NFT can be minted at any time, as long as the position is open. When the position is open and then the collateral is blacklisted, all positions using it will not be able to mint an NFT. This seems to be over-restrictive, as NFT can be minted any time while the position is opened.

Impact

NFT can't be minted for collateral that is blacklisted after the position is opened.

Attack Scenario

1. Bob creates a position with USDT as collateral and doesn't mint an NFT for it yet.
2. The admin blacklists USDT. No new positions with this collateral token can be made.
3. Bob's position is earning a lot. He decides to mint the NFT and borrow it on a lending market.
4. Bob cannot mint the NFT, because of the check described above.

Location of Affected Code

File: src/NFT/NftHandler.sol#L112-L114

Recommendation

Remove blacklisted collateral check and allow NFT to be minted for any open position.

Team Response

Acknowledged and fixed as suggested.

[L-06] Mint Fee is Not Aligned With the Docs and Also Big Fees Can be Avoided

Severity

Low Risk

Description

As stated in the docs the mint fee is `1$` at max. However, the implementation takes `0,05%` of the position size.| of the position size.| of the position size.

If the intention is to implement the fee percentage as above, this can also be circumvented by below:

The user opens a position with min amounts (small size). The user mints the position NFT (percentage of the small size yields to small fee). The user increases the position to avoid big mint fees at the beginning.

Impact

1. The users might be overcharged than the declared.
2. The intended mechanism can be gamed to avoid paying larger fees.

Location of Affected Code

File: src/NFT/NftHandler.sol#L118

```
/// @inheritdoc INFTHandler
function mintNFT(
  address adapter,
  address to
) external override returns (uint256 tokenId) {
// code

  address platformLogic = comptroller.getPlatformLogic();
  uint256 amount = ((positionsData[1] + positionsData[10]) * 1e6) / 1e30;

  PairPositionData memory pairData = IPearGmxFactory(gmxFactory)
    .getPairPositionData(adapter);
  address collateralToken = pairData.short.collateralToken;
  IERC20 _token = IERC20(collateralToken);

  if (comptroller.allowedTokens(_token) != true) {
    revert NFTHandler_TokenNotAllowed();
  }

  IPlatformLogic(platformLogic).applyMintFeeErc20Gmx(
    msg.sender,
    amount, //@audit The fee is taken as per the position size
    _token,
    adapter
  );

// code
}
```

Recommendation

It's recommended to apply fees as stable amounts.

Team Response

Acknowledged and fixed as suggested.

[L-07] No Possibility to Close Liquidated Trade

Severity

Low Risk

Description

When a trade is liquidated, GMX doesn't send any callback. The position in Pear is still visible as open, however, no action can be done for it, because of checks for 0 positions. The position can't be increased, because the fee calculated based on the current position will be 0, and will revert in the `PlatformLogic.calculateFees()` function:

```
function calculateFees(
  uint256 _amount,
  uint256 _bps
) public pure override returns (uint256) {
/// @notice Check that the amount multiplied by the Basis Points is
   greater than or equal to 10000
// This ensures that we're not running into the issue of values being
   rounded down to 0.
  if ((_amount * _bps) < 10000) revert PlatformLogic_WrongFeeAmount();
  return (_amount * _bps) / 10000;
}
```

Impact

Position state not updated in case of account liquidation.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L477-L691

Recommendation

Consider adding the following code to `GmxFasctory.closePositions()`:

```
function closePositions(
  ClosePositionsArgs memory args
)
  external
  payable
  override
  returns (
    address adapter,
    bytes32 longPositionId,
    bytes32 shortPositionId
  )
{
```

```
  if (msg.sender != adapterOwners[args.adapter]) {
    revert Errors.GmxFactory_NotAdapterOwner();
  }

  if (
    positionDetails[args.adapter][msg.sender] != PositionStatus.Opened
  ) {
    revert Errors.GmxFactory_PositionNotOpened();
  }

  uint256[] memory data = getPosition(args.adapter);
```

```
+ if (data[1] + data[10] == 0 &&
+   positionDetails[args.adapter][msg.sender] != PositionStatus.Opened)
+ {
+   positionDetails[args.adapter][msg.sender] = PositionStatus.Closed;
+   adapter = args.adapter;
+   return;
+ }
}
```

However, even with this approach, if the user doesn't call the function, the position will still be visible as Opened. Hence, consider introducing automatic or manual means to change the status of the empty opened position to close.

Team Response

Acknowledged and fixed as suggested.

[L-08] Functions Need To Be Called Only Once

Severity

Low Risk

Description

There is a set of functions that need to be called only once due to the nature of the contracts the Pear is serving. Accordingly, the protocol could proceed to an irreversible state if called more than once.

Below are the functions that require to be locked after being called by the deployer address.

1. Calling `setNftHandler()` will re-set the contract address.

File: **`Comptroller.sol`**

```
/// @inheritdoc IComptroller
function setNftHandler(
  address _nftHandler
) external override onlyAdmin notZeroAddress(_nftHandler) {
  emit SetNftHandler(nftHandler, _nftHandler);

  nftHandler = _nftHandler;
}
```

`NFTHandler.sol` has the `mintNFT()` function to mint a position to obtain ownership of the position. It checks whether the ownership is claimed by checking the mapping on **L:90-92**, if false, accordingly, it proceeds to mint the position unless the position is closed.

File: **NftHandler.sol**

```
/// @inheritdoc INFTHandler
function mintNFT(
    address adapter,
    address to
) external override returns (uint256 tokenId) {
  address gmxFactory = comptroller.getGmxFactory();

  address adapterOwner = IPearGmxFactory(gmxFactory).adapterOwners(
    adapter
  );

  if (adapterOwner != msg.sender) {
    revert NftHandler_NotPositionOwner();
  }

  if (mintedPositions[adapter]) {
    revert NftHandler_PositionAlreadyMinted();
  }
// code
}
```

If the contract is re-set, since the mappings are not synced, users who have positions can reclaim their positions by re-minting their positions. This would double the position's claims while the protocol should only have half.

2. `setPoisitionNft()` function sets or changes the `PoisitionNFT.sol` address:

File: **Comptroller.sol**

```
/// @inheritdoc IComptroller
function setPositionNft(
  address _positionNft
) external override onlyAdmin notZeroAddress(_positionNft) {
  emit SetPositionNft(positionNft, _positionNft);

  positionNft = _positionNft;
}
```

as well as ERC721 contract address due to below:

File: `PositionNFT.sol`

```
contract PositionNFT is IPositionNFT, -->>ERC721<<-- {
```

So the actions `transferNFT()` and `burnNft()` will be bricked due to `L:142 and L:167`:

File: `NftHandler.sol`

```
/// @inheritdoc INFTHandler
function burnNFT(uint256 tokenId) external override {
  address gmxFactory = comptroller.getGmxFactory();
  address positionNft = comptroller.getPositionNft();

> if (IPositionNFT(positionNft).ownerOf(tokenId) != msg.sender) {
    revert NftHandler_NotNftOwner();
  }
// code
}
```

```
/// @inheritdoc INFTHandler
function transferNft(uint256 tokenId, address to) external override {
  address gmxFactory = comptroller.getGmxFactory();
  address positionNft = comptroller.getPositionNft();

> if (IPositionNFT(positionNft).ownerOf(tokenId) != msg.sender) {
    revert NftHandler_NotNftOwner();
  }
// code
}
```

3. Calling `setBackendVerifierAddress()` will invalidate the existing signatures and `createReferralCode()` will revert:

File: `PlatformLogic.sol`

```
/// @inheritdoc IPlatformLogic
function setBackendVerifierAddress(
  address _newBackendVerifier
) external override onlyAdmin notZeroAddress(_newBackendVerifier) {
  emit BackendVerifierChanged(backendVerifier, _newBackendVerifier);
  backendVerifier = _newBackendVerifier;
}
```

File: `PlatformLogic.sol`

```
if (signer != backendVerifier) revert PlatformLogic_InvalidSigner();
```

However, calling this function is way less severe due to the turnaround it carries.

Location of Affected Code

File: src/Comptroller.sol#L148-L154

File: src/Comptroller.sol#L166-L172

File: src/PlatformLogic.sol#L164-L169

Recommendation

We recommend adding a lock to these functions like the `initializer` modifier.

Team Response

Acknowledged and fixed as suggested.

[L-09] The `CalculateFees()` Function Will Revert If Any Low Decimal Token Is Allowed as Fee Token

Severity

Low Risk

Description

`calculateFees()` function is prone to revert due to `L:282` in case a low decimal token is allowed to be used as a fee token like GUSD (2 decimals) .

Location of Affected Code

File: src/PlatformLogic.sol#L282

```
/// @inheritdoc IPlatformLogic
function calculateFees(
  uint256 _amount,
  uint256 _bps
) public pure override returns (uint256) {
/// @notice Check that the amount multiplied by the Basis Points is
   greater than or equal to 10000
// This ensures that we're not running into the issue of values being
   rounded down to 0.
  if ((_amount * _bps) < 10000) revert PlatformLogic_WrongFeeAmount();
  return (_amount * _bps) / 10000;
}
```

Recommendation

1. Consider adding a minimum fee safeguard against setting the fees too low.
2. Correcting the fee amount considering the token decimals would be a good approach for future versions if any low decimal tokens are to be utilized.

Team Response

Acknowledged.

[L-10] Centralization Risks

Severity

Low Risk

Description

While the Pear protocol requires a trusted party to adjust to market conditions and possible updates of the code, some of the functionalities allow compromised admin accounts to exploit the protocol.

Pear treasury and staking contracts may censor user's Vertex transactions

ETH fees in Vertex can be reverted by Treasury contract, or PearStaking contract, allowing the admin to censor users:

Setting 100% fees

The fees can be set as high as 100%, which means that the treasury can receive all fees.

Setting malicious contracts in Comptroller

The comptroller holds a reference to all Pear, GMX and Vertex contracts. Admin can change the core contracts, as well as change the Comptroller implementation in all of the contracts. Because some of them deal directly with user funds, changing them into a malicious contract, and stealing from all protocol users.

Impact

Complete protocol dysfunction and users' funds lost in case of admin account compromise.

Location of Affected Code

File: src/PlatformLogic.sol#L889

```
function _applyPlatformFeeEthVertex(
  address _referee,
  uint256 _grossAmount
) internal {
// code
```

```
  (bool _successTreasury, ) = PearTreasury.call{
    value: _amountToBeSendToTreasury
  }("");

  if (!_successTreasury) revert PlatformLogic_TransactionFailed();

  uint256 _amountToBeSendToStakers = _feeAmount -
    _amountToBeSendToTreasury;

  if (_amountToBeSendToStakers > 0) {
    (bool _successStakers, ) = PearStakingContract.call{
      value: _amountToBeSendToStakers
    }("");

    if (!_successStakers) revert PlatformLogic_TransactionFailed();
  }
// code
}
```

Recommendation

For each of the following issues:

1. Make sure that the admin account is Timelock contract operated via a multi-sig contract
2. For each of the points listed above, try to minimize the impact of possible admin account compromise

Team Response

Acknowledged and fixed as suggested.

[L-11] `FeeSplit` Calculation Might Truncate to Zero

Severity

Low Risk

Description

GmxFactory's `createIncreasePositions()` & `createDecreasePositions()` functions calculate the `feeSplit` as:

```
feeSplit = (args.amountInLong * 10000) / amount;
```

Since `amount` value is `args.amountInLong + args.amountInShort`, there's a possibility that `feeSplit` calculation might truncate to zero.

Location of Affected Code

File: src/Factory/GmxFactory.sol#L287
File: src/Factory/GmxFactory.sol#L399

```
feeSplit = (args.amountInLong * 10000) / amount;
```

Recommendation

Recommend checking `feeSplit != 0`.

Subject to design choice:

- The function can revert.
- The function can implement minimum fees (a new variable).

Team Response

Acknowledged and fixed as suggested.

shieldify

Thank you!