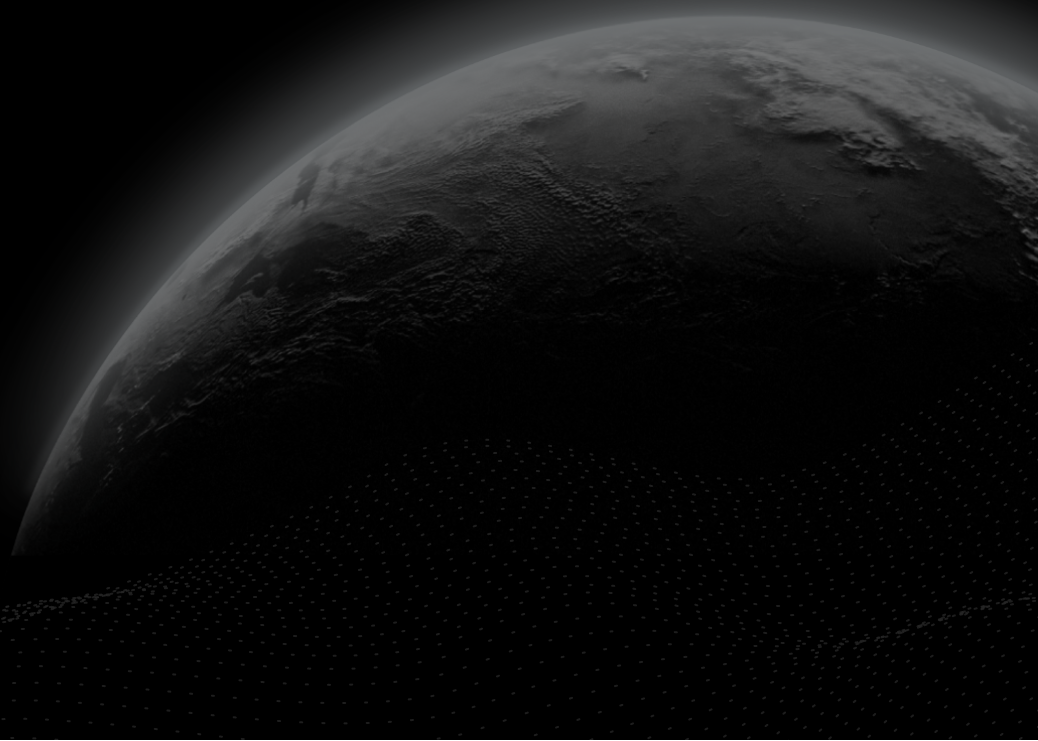
CERTIK

Security Assessment

# Jungle Exchange

CertiK Assessed on Jun 4th, 2024




CertiK Assessed on Jun 4th, 2024

# Jungle Exchange

The security assessment was prepared by CertiK, the leader in Web3.0 security.

## Executive Summary

| TYPES | ECOSYSTEM | METHODS |
|---|---|---|
| DeFi | Ethereum (ETH) | Manual Review, Static Analysis |

| LANGUAGE | TIMELINE | KEY COMPONENTS |
|---|---|---|
| Solidity | Delivered on 06/04/2024 | N/A |

CODEBASE
jungle-synthetics
View All in Codebase Page

COMMITS
87e36f11f12b750b5613cb514357ddc664d70392
View All in Codebase Page

## Vulnerability Summary

| 23 | 9 | 0 | 0 | 14 | 0 |
|---|---|---|---|---|---|
| Total Findings | Resolved | Mitigated | Partially Resolved | Acknowledged | Declined |

| | | | |
|---|---|---|---|
| 0 | Critical | | Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
| 1 | Major | 1 Acknowledged | Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project. |
| 10 | Medium | 5 Resolved, 5 Acknowledged | Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform. |
| 12 | Minor | 4 Resolved, 8 Acknowledged | Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions. |
| 0 | Informational | | Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

# TABLE OF CONTENTS | JUNGLE EXCHANGE

## Disclaimer

# CODEBASE | JUNGLE EXCHANGE

## Repository

jungle-synthetics

## Commit

87e36f11f12b750b5613cb514357ddc664d70392

# AUDIT SCOPE | JUNGLE EXCHANGE

18 files audited ● 7 files with Acknowledged findings ● 2 files with Resolved findings ● 9 files without findings

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| JUN | jungle-official/jungle-synthetics | core/Jungle.sol | e88e7a2f89c977deff62ca09e8ac45ed1fc49c054f4c5847b060c15d81daf5fb |
| JSB | jungle-official/jungle-synthetics | core/JungleSetting.sol | 0b69536a3714f4c70850d1b7d1c94df3e8d51822f0d5b8e118a8fc7790de954b |
| LPM | jungle-official/jungle-synthetics | core/LPManager.sol | ece1b5ae7c1c57c718937d4e1bff8c6d5a27a5aea84372f39d7042e56b1b3a7f |
| LPA | jungle-official/jungle-synthetics | core/LPMarket.sol | 458432417a3571593215006682c9b17cdc250411ccc08b7daacab375a47cdcd8 |
| MTB | jungle-official/jungle-synthetics | core/MarketTick.sol | e5dba276cf8aa4892e44742b56eb7291baed80e39409ce5402ad5ddedb195930 |
| PFB | jungle-official/jungle-synthetics | core/PriceFeed.sol | 577ce7b17ca07e2773b9dae0e7e9f8b8e7edf8d0e3c5bee790b00463601583b5 |
| VAU | jungle-official/jungle-synthetics | core/Vault.sol | 9306e03c85e914d7a7ee4a91b40559dbd9a365bb4b7d8af537995680a94db4d8 |
| JRB | jungle-official/jungle-synthetics | periphery/JungleReader.sol | 25e34b4a4fe6e750ac2e8e4d9e1258491aa13d93d2968f9e970a28af0eb343d3 |
| LPR | jungle-official/jungle-synthetics | periphery/LPReader.sol | c3500fd2bf218f95aba7ed30556b49cf11b121afca93a3e9b82a13ce28571c63 |

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| JUS | jungle-official/jungle-synthetics | core/JUSD.sol | a990ab4134dfefe2c98c2cb19c748757c674474a3e05432bb8d388b9504ccd02 |
| LPD | jungle-official/jungle-synthetics | core/LPMarketDeployer.sol | 0809e7b1fc2fdf72696a01bb709bd15d81f94e859652a1a45a24439ca5271b0e |
| LPT | jungle-official/jungle-synthetics | core/LPTickDeployer.sol | a4d101b656b54ce52b1345a42b8b76682ff9978c4467b2d07afced92ea39a33a |
| BMB | jungle-official/jungle-synthetics | core/library/BitMath.sol | 6a7de242ba91281bef281e4dacc83736755152b85d8522a18a18728858d09ea9 |
| LMA | jungle-official/jungle-synthetics | core/library/LPMarketAddr.sol | bb9ad8be75521846ff401b74ab87a8a548bc12c1952957d973ef5993bbcf3873 |
| POS | jungle-official/jungle-synthetics | core/library/Positions.sol | c82814d482091d0d991f64b1db25ceba9806cffa380bab64354e97d554997509 |
| TBB | jungle-official/jungle-synthetics | core/library/TickBitmap.sol | 35bd83052e34cdfa4da579e5b2202a898ba35b54885e0b7b35ab25f066e829f0 |
| TMB | jungle-official/jungle-synthetics | core/library/TickMath.sol | 498f76536e80d5c27fb785e7784e83482472bbd4196389603a5fea7ccb42ae06 |
| CON | jungle-official/jungle-synthetics | periphery/Constants.sol | c82096322759d97495ea9e8b4def8453de0fc53c151bf074c98fc861494160b4 |

# APPROACH & METHODS | JUNGLE EXCHANGE

This report has been prepared for Jungle Exchange to discover issues and vulnerabilities in the source code of the Jungle Exchange project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# FINDINGS | JUNGLE EXCHANGE

| 23 | 0 | 1 | 10 | 12 | 0 |
|---|---|---|---|---|---|
| Total Findings | Critical | Major | Medium | Minor | Informational |

This report has been prepared to discover issues and vulnerabilities for Jungle Exchange. Through this audit, we have uncovered 23 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| **COR-04** | **Centralization Related Risks** | **Centralization** | **Major** | **Acknowledged** |
| COR-01 | No Upper Limit For Fee | Logical Issue | Medium | Resolved |
| COR-02 | Variable Usage Before Setting Value | Logical Issue | Medium | Acknowledged |
| JSB-01 | Incorrect Parameter Usage In `validatePosition` Function | Logical Issue | Medium | Acknowledged |
| JSB-02 | Incorrect Handling Of Decreased Position In `decreaseOpenInterest` Function | Logical Issue | Medium | Resolved |
| JUN-02 | Unchecked Return Value In `isClose` Function | Logical Issue | Medium | Acknowledged |
| JUN-03 | Unauthorized Cancellation Of Keeper's Orders In `cancelLimitOrder` Function | Logical Issue | Medium | Resolved |
| LPA-03 | Inconsistent Access Control Leading To Function Unavailability | Inconsistency, Access Control | Medium | Resolved |
| LPA-07 | The Variable `poolStates` Is Not Updated When Performing Liquidation | Logical Issue | Medium | Acknowledged |
| LPM-01 | Lack Of Access Control | Access Control | Medium | Resolved |
| PFB-01 | Missing Validation On The Return Value Of Oracle | Volatile Code | Medium | Acknowledged |

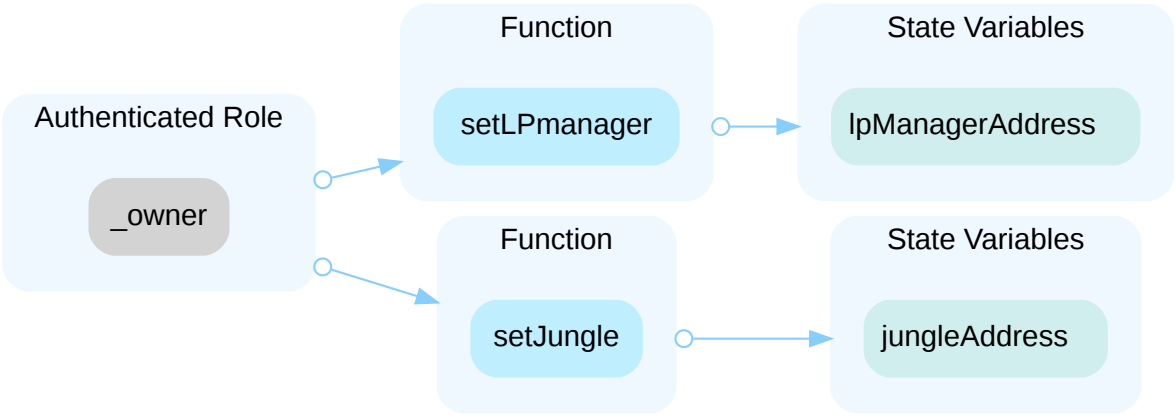
| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| CON-01 | Possible Integer Overflow And Dead Loop | Logical Issue | Minor | Resolved |
| COR-03 | Question Related To Bid-Ask Spread | Logical Issue | Minor | Resolved |
| COR-05 | Contradictory Implementation In Matching Mechanism | Design Issue | Minor | Acknowledged |
| GLOBAL-01 | Missing Implementation Of Individual Pool Priority | Design Issue | Minor | Acknowledged |
| JUN-04 | The Surplus Native Tokens Are Not Returned | Design Issue | Minor | Acknowledged |
| JUN-05 | Potential Orders Filled At Worse Prices | Logical Issue | Minor | Acknowledged |
| LPA-06 | Potential Logical Inconsistencies `liquidatePosition` And `adl` | Logical Issue | Minor | Acknowledged |
| LPA-08 | Potential Average Price Is Not Correct | Logical Issue | Minor | Resolved |
| LPM-02 | Potential Allow Unauthorized Withdrawals In `claimReward` Function | Logical Issue | Minor | Acknowledged |
| LPM-03 | Incorrect Boolean Expression | Logical Issue | Minor | Acknowledged |
| MTB-01 | Missing Service Fee Handling Mechanism | Design Issue | Minor | Resolved |
| PFB-02 | Third-Party Dependency Usage | Design Issue | Minor | Acknowledged |

# COR-04 | CENTRALIZATION RELATED RISKS

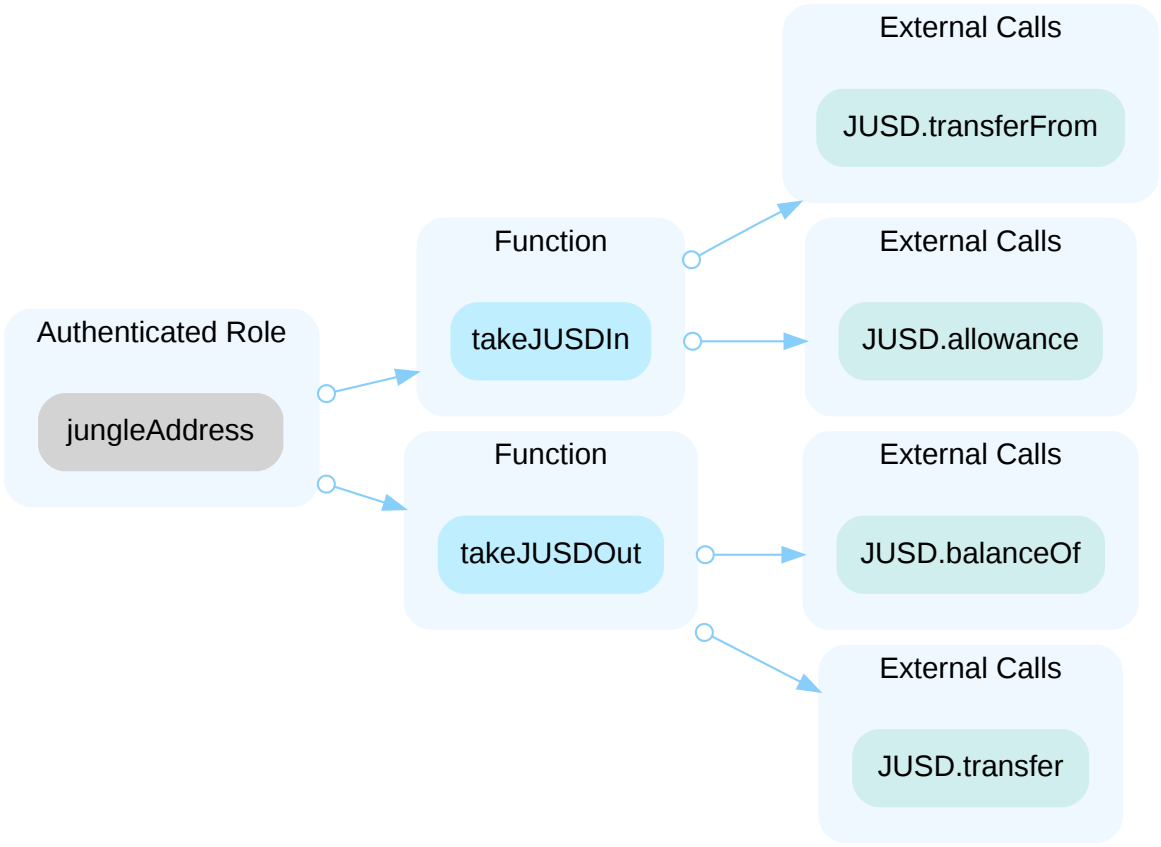
| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization | Major | core/Jungle.sol: 74, 78, 267, 651, 751; core/JungleSetting.sol: 94, 98, 103, 108, 114, 125, 131, 135, 140, 149, 446, 459, 492, 514; core/LPManager.sol: 75, 79; core/LPMarket.sol: 107, 112, 140, 144, 181, 243, 958, 975, 996, 1027; core/MarketTick.sol: 96, 102, 106, 345, 420, 638, 645, 663, 797, 808, 835, 880, 898, 921; core/PriceFeed.sol: 22; core/Vault.sol: 33, 37, 41, 46 | Acknowledged |

## Description

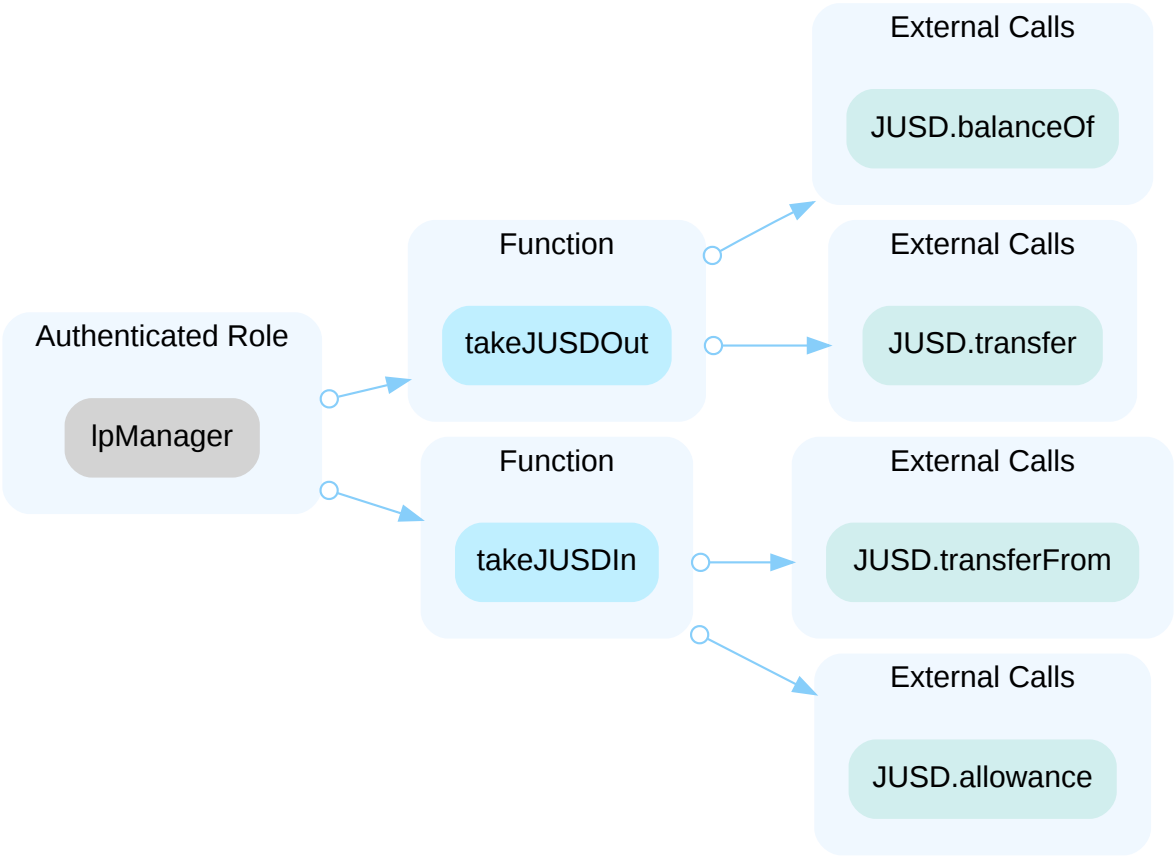
In the contract `Vault` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set `Jungle` and `LPManager` address.



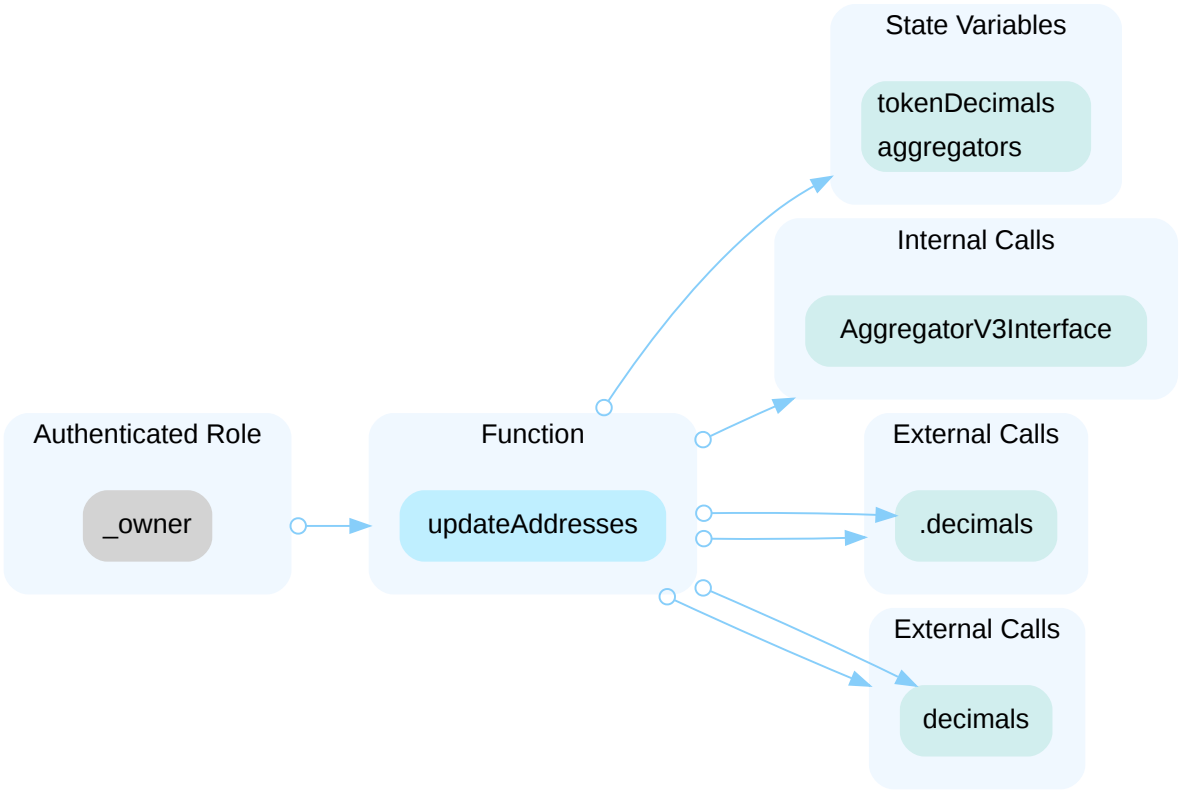

In the contract `Vault` the role `jungleAddress` has authority over the functions shown in the diagram below. Any compromise to the `jungleAddress` account may allow the hacker to take advantage of this authority and transfer in/out JUSD to/from the vault.



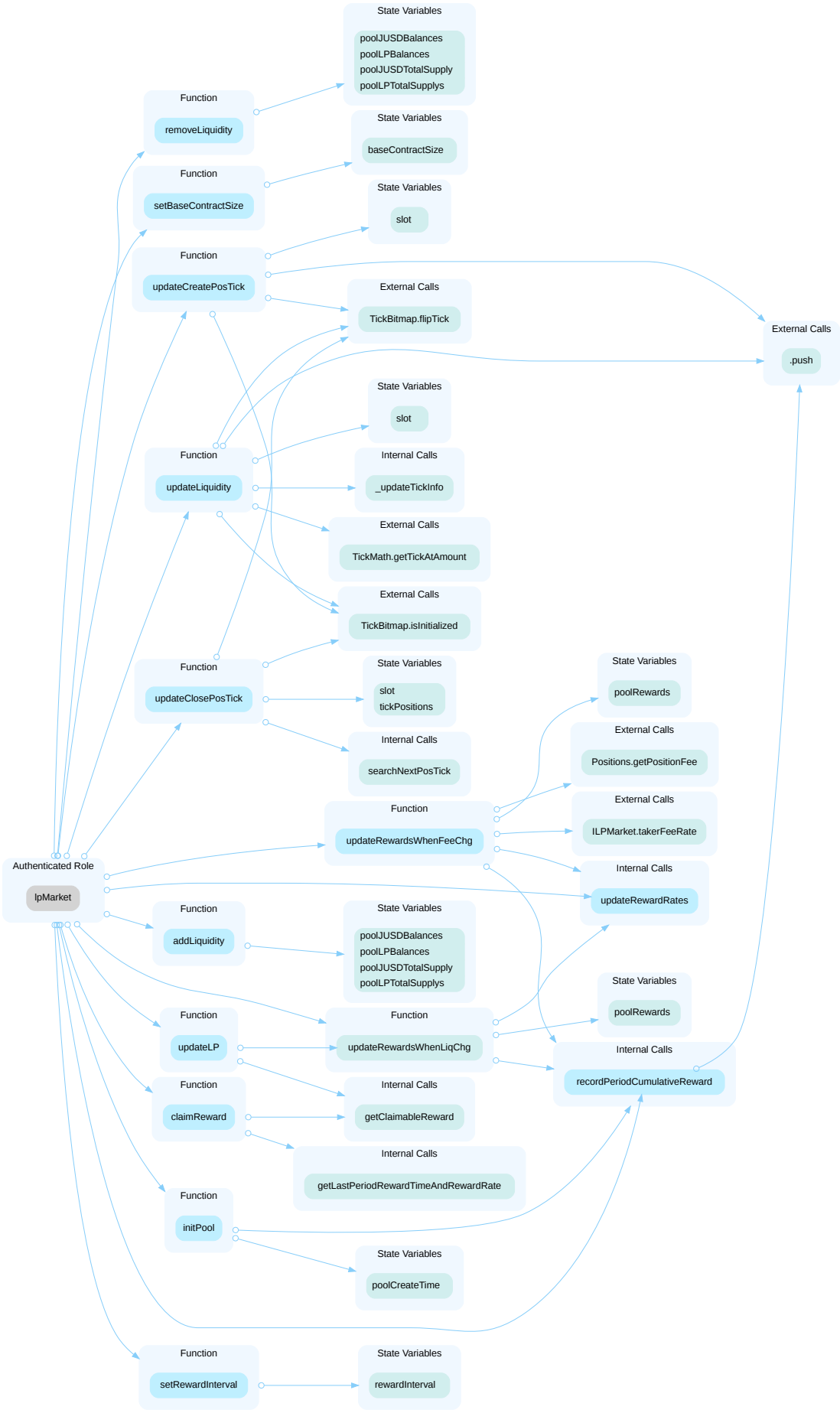

In the contract `Vault` the role `lpManager` has authority over the functions shown in the diagram below. Any compromise to the `lpManager` account may allow the hacker to take advantage of this authority and transfer in/out JUSD to/from the vault.



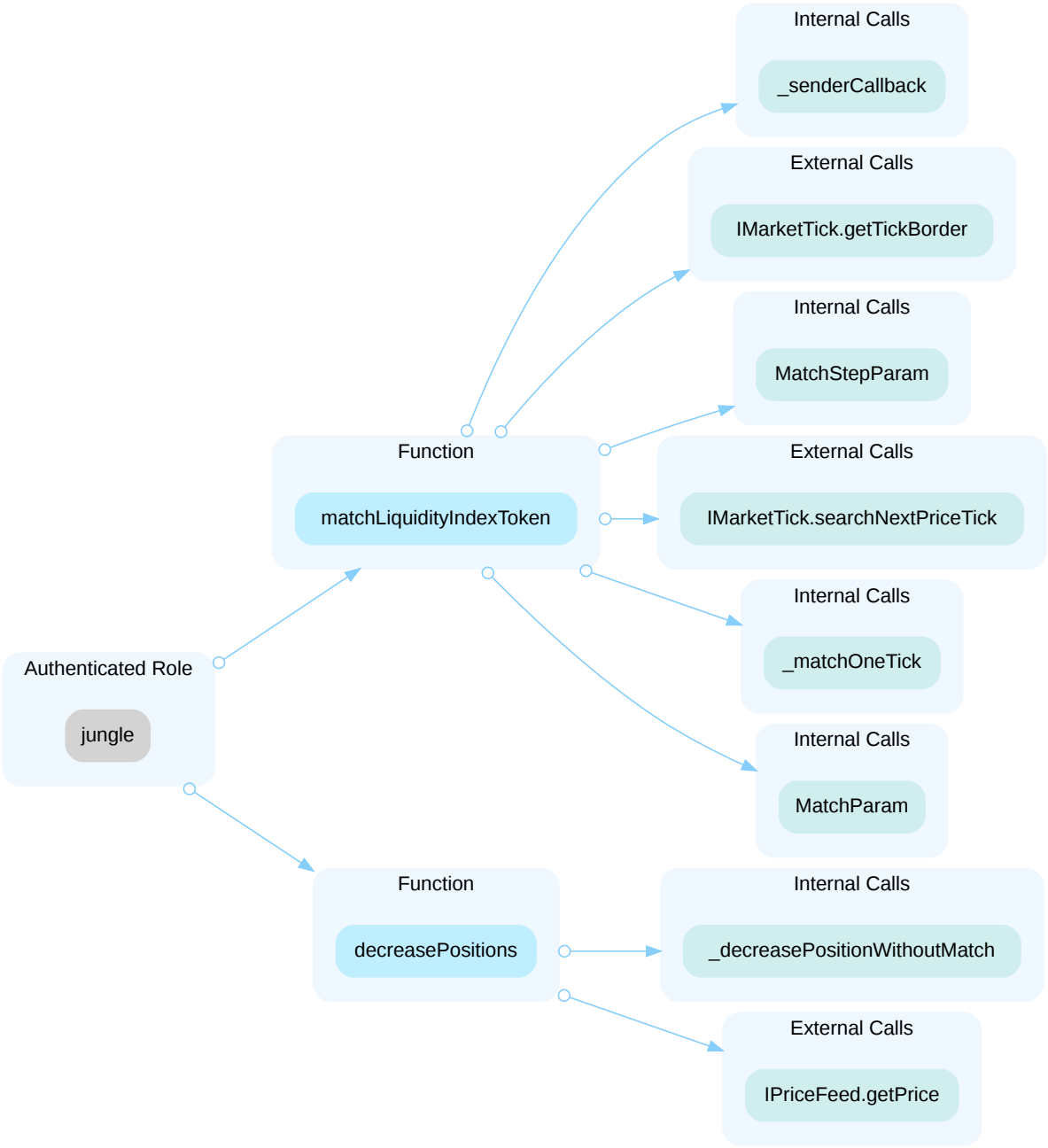
In the contract `PriceFeed` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set `aggregators` and `tokenDecimals`.




In the contract `MarketTick` the role `lpMarket` has authority over the functions shown in the diagram below. Any compromise to the `lpMarket` account may allow the hacker to take advantage of this authority, setting significant parameters and invoking critical functions.


State Variables
poolJUSDBalances
poolLPBalances
poolJUSDTotalSupply
poolLPTotalSupplys
Function
removeLiquidity
State Variables
baseContractSize
Function
setBaseContractSize
State Variables
slot
Function
updateCreatePosTick
External Calls
TickBitmap.flipTick
External Calls
.push
State Variables
slot
Function
updateLiquidity
Internal Calls
_updateTickInfo
External Calls
TickMath.getTickAtAmount
External Calls
TickBitmap.isInitialized
Function
updateClosePosTick
State Variables
slot
tickPositions
State Variables
poolRewards
External Calls
Positions.getPositionFee
Internal Calls
searchNextPosTick
Function
updateRewardsWhenFeeChg
External Calls
ILPMarket.takerFeeRate
Internal Calls
updateRewardRates
Authenticated Role
lpMarket
Function
addLiquidity
State Variables
poolJUSDBalances
poolLPBalances
poolJUSDTotalSupply
poolLPTotalSupplys
State Variables
poolRewards
Function
updateLP
Function
updateRewardsWhenLiqChg
Internal Calls
recordPeriodCumulativeReward
Function
claimReward
Internal Calls
getClaimableReward
Internal Calls
getLastPeriodRewardTimeAndRewardRate
Function
initPool
State Variables
poolCreateTime
Function
setRewardInterval
State Variables
rewardInterval

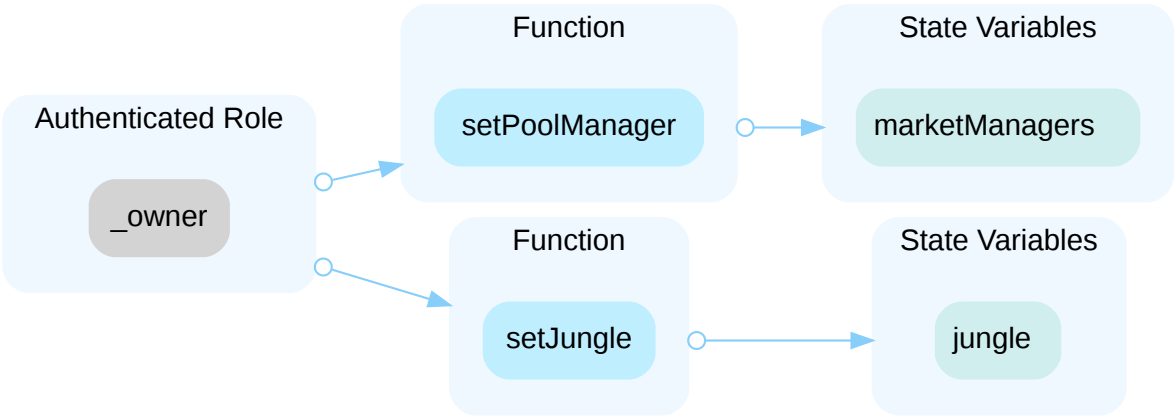
In the contract `LPMarket` the role `jungle` has authority over the functions shown in the diagram below. Any compromise to the `jungle` account may allow the hacker to take advantage of this authority.



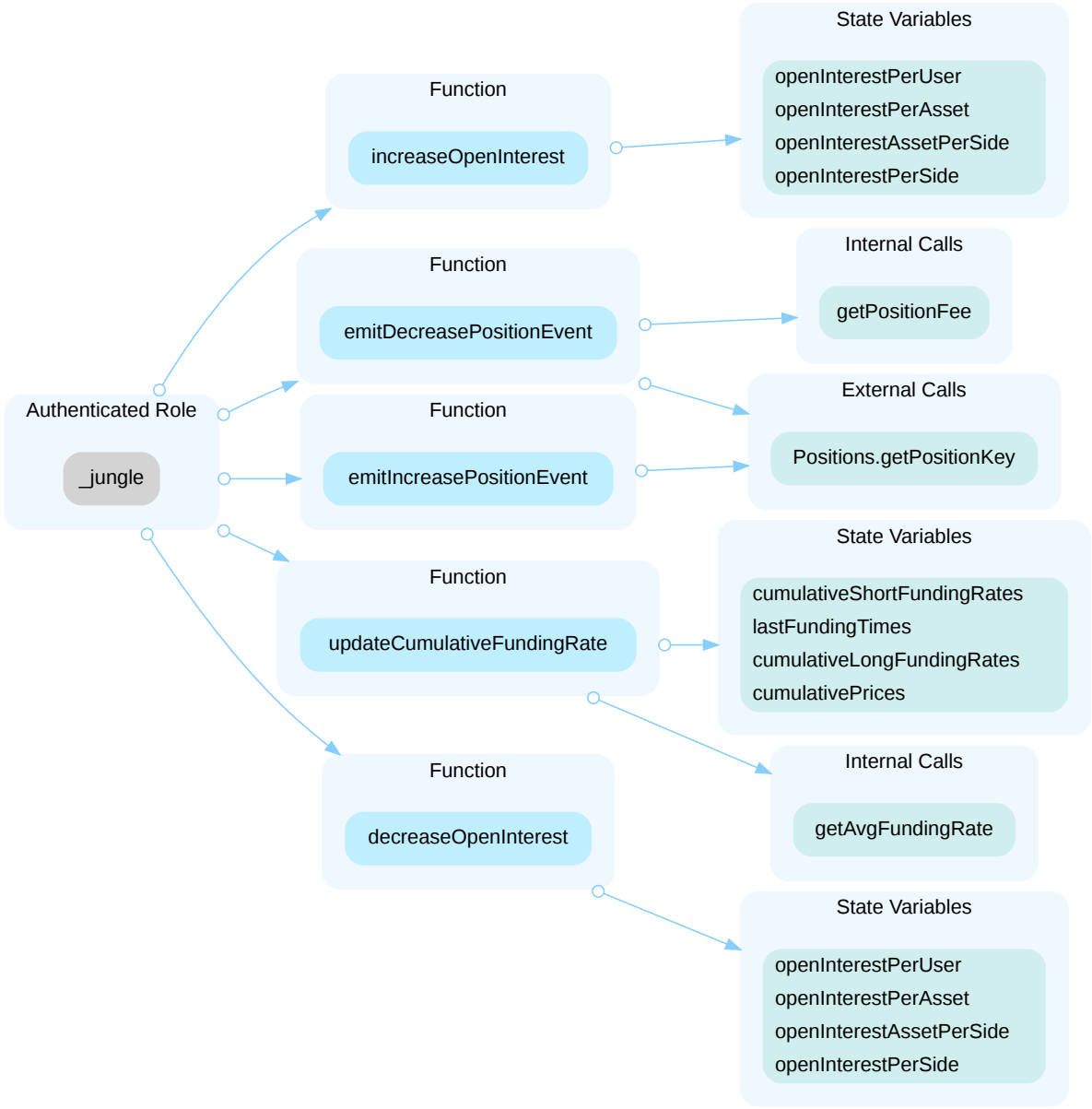

In the contract `LPMarket` the role `lpManager` has authority over the functions shown in the diagram below. Any compromise to the `lpManager` account may allow the hacker to take advantage of this authority and set `jungle`, `poolState`, `takerFeeRate`, and invoke critical functions.


External Calls
IMarketTick.setBaseContractSize
State Variables
baseContractSize
marketName
strategies
Function
setPool
External Calls
IMarketTick.initPool
Internal Calls
getRemoveLiquidityParam
Internal Calls
_updatePoolTick
Function
removeLiquidity
External Calls
IMarketTick.removeLiquidity
External Calls
IMarketTick.poolLPTotalSupplys
External Calls
IMarketTick.updateLP
Function
addLiquidity
External Calls
IMarketTick.addLiquidity
Internal Calls
getNetValue
External Calls
IMarketTick.searchNextPriceTick
Internal Calls
_matchOneTick
Internal Calls
MatchParam
Function
matchLiquidityIndexToken
Internal Calls
_senderCallback
External Calls
IMarketTick.getTickBorder
Internal Calls
MatchStepParam
Internal Calls
validateLiquidation
Function
liquidatePosition
Internal Calls
_liquidatePosition
Authenticated Role
lpManager
Function
adl
External Calls
IMarketTick.validateLiquidation
External Calls
IPriceFeed.getPrice
Function
decreasePositions
Internal Calls
_decreasePositionWithoutMatch
Function
claimReward
External Calls
IMarketTick.claimReward
Function
setTakerFeeRate
State Variables
takerFeeRate
Function
setJungle
State Variables
jungle

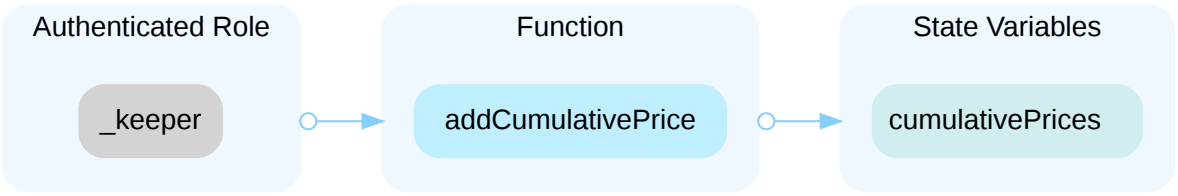
In the contract `LPManager` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set `marketManagers[_indexToken][_manager]`.



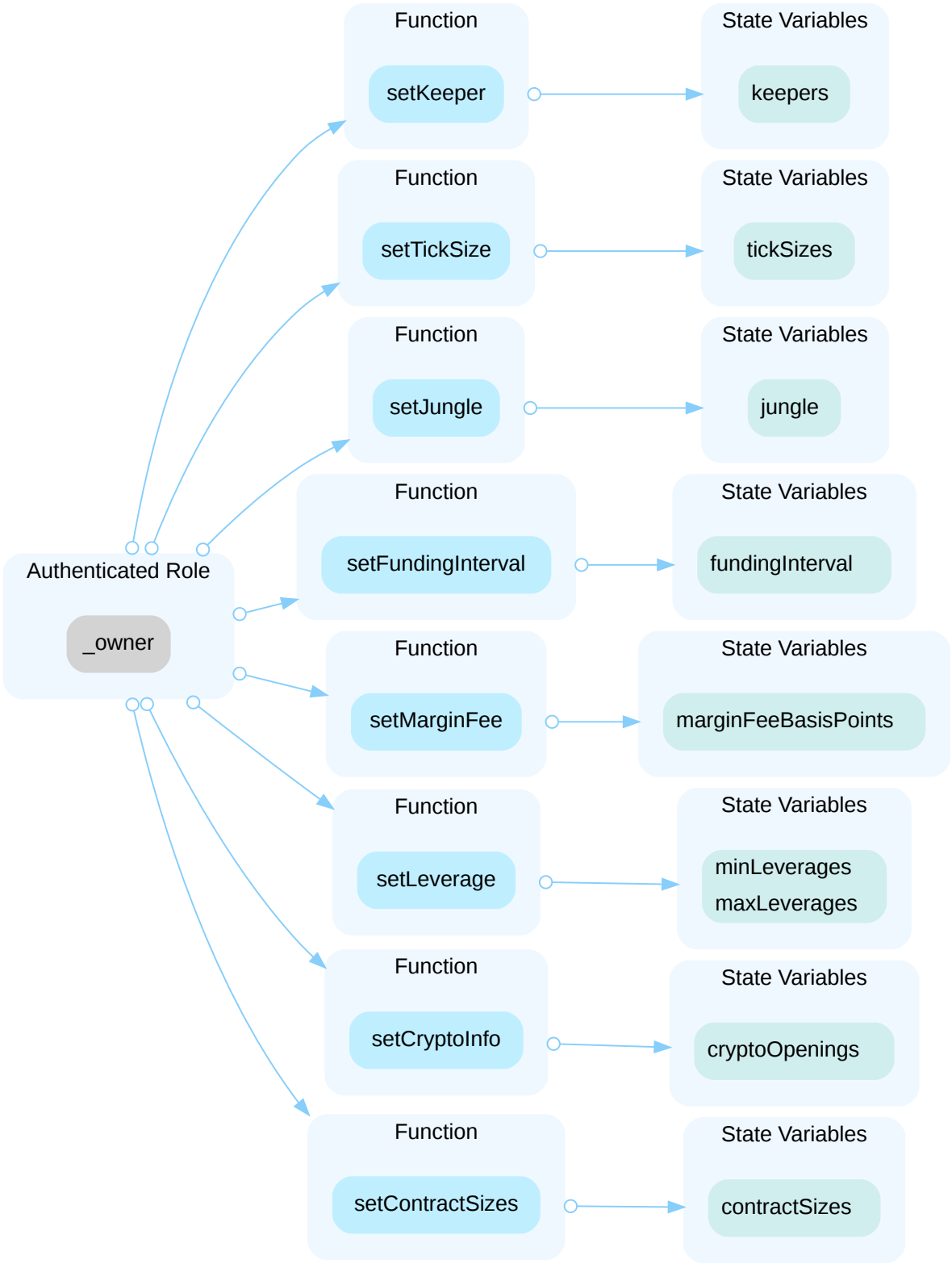

In the contract `JungleSetting` the role `_jungle` has authority over the functions shown in the diagram below. Any compromise to the `_jungle` account may allow the hacker to take advantage of this authority, increasing/decreasing open interest, changing cumulative funding rate.



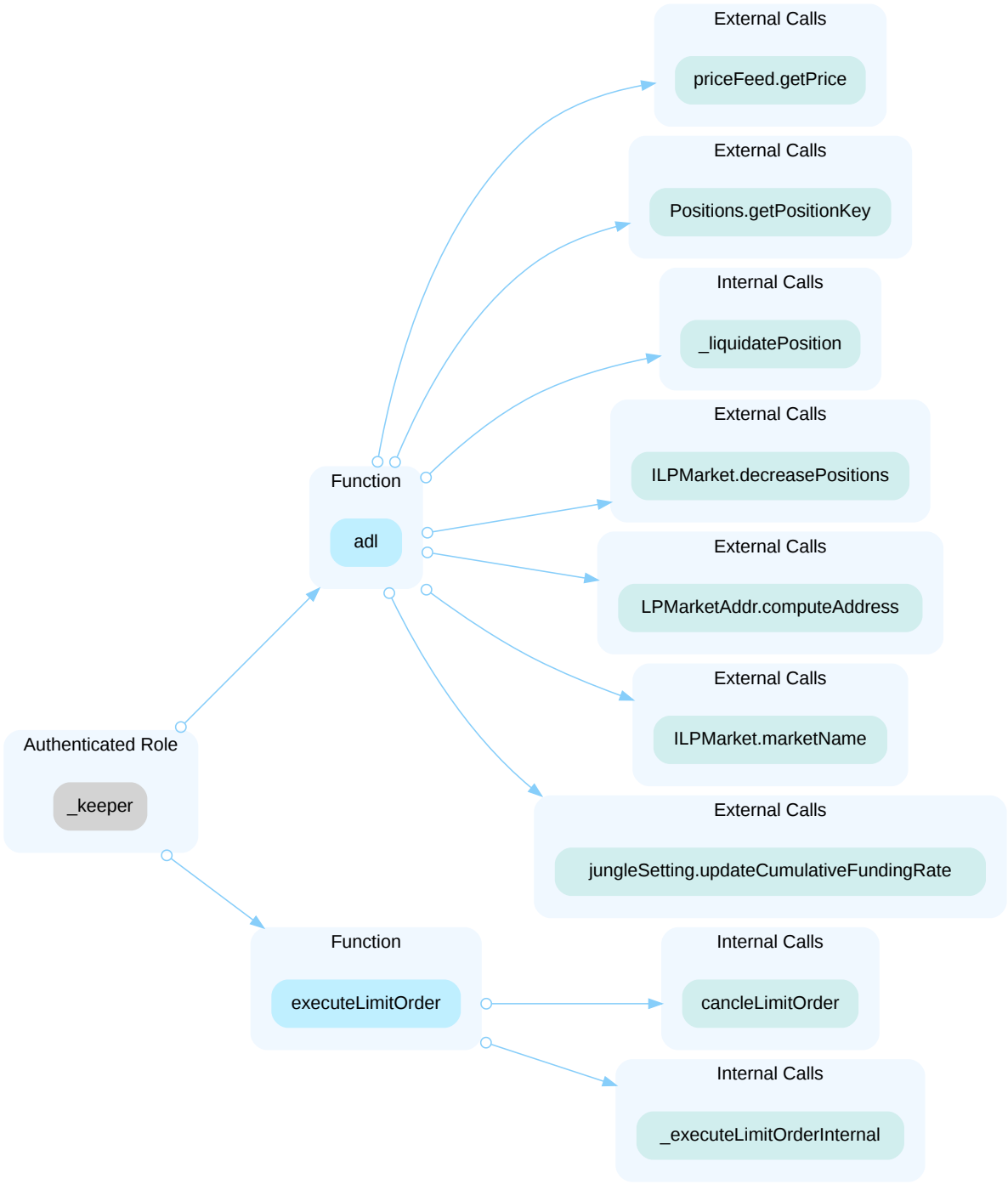

In the contract `JungleSetting` the role `_keeper` has authority over the functions shown in the diagram below. Any compromise to the `_keeper` account may allow the hacker to take advantage of this authority and increase `cumulativePrices`.



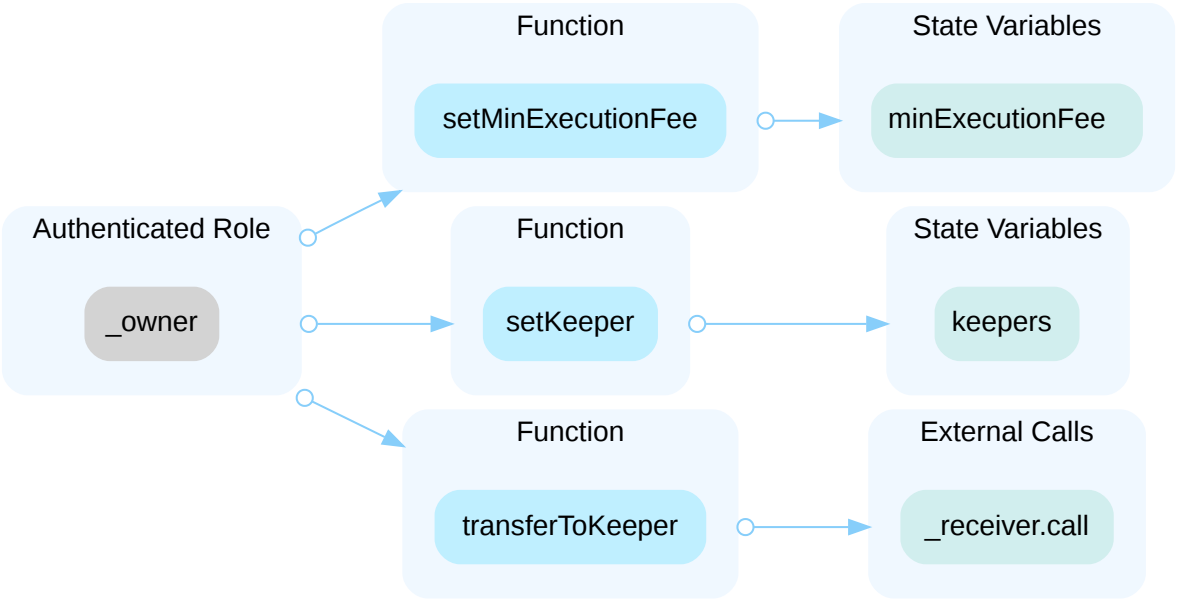

In the contract `JungleSetting` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set keepers, `jungle`, `marginFeeBasisPoints` and other significant global variables.




In the contract `Jungle` the role `_keeper` has authority over the functions shown in the diagram below. Any compromise to the `_keeper` account may allow the hacker to take advantage of this authority, triggering auto-deleveraging and executing limit orders.




In the contract `Jungle` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set `minExecutionFee` and `keepers`.




## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

### Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

### Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

### Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: Issue acknowledged. I won't make any changes for the current version.

We will consider Timelock and multi-signature in our contracts.

**[CertiK, 04/07/2024]**: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

# COR-01 | NO UPPER LIMIT FOR FEE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/Jungle.sol: 74~75; core/LPMarket.sol: 140~141 | Resolved |

## Description

There are no upper boundaries for function `setMinExecutionFee` and `setTakerFeeRate`, which is used to set `minExecutionFee` and `takerFeeRate`. It is possible to set the total fee rate up to any arbitrary amount. In the contract `Jungle` the role `_owner` has authority over the function `setMinExecutionFee`. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set the total fee rate up to any arbitrary amount.

In the contract `LPMarket` the role `lpManager` has authority over the function `setTakerFeeRate`. Any compromise to the `lpManager` account may allow the hacker to take advantage of this authority and set the total fee rate up to any arbitrary amount.

## Recommendation

Introduce a maximum fee threshold in the function to ensure fee values remain within acceptable limits. This safeguard will provide predictability and fairness in fee-related operations.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: 558d3ede2271c792754c595c7da7c73f73f70e6b.

# COR-02 | VARIABLE USAGE BEFORE SETTING VALUE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/JungleSetting.sol: 23, 36, 42; core/MarketTick.sol: 30, 31, 32 | Acknowledged |

## Description

This issue pertains to a situation in which a contract utilizes a variable before setting its value, leading to potential errors and unexpected behavior in the contract's execution.

## Recommendation

We recommend setting a suitable value before using the state variable.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team acknowledged the finding.

# JSB-01 INCORRECT PARAMETER USAGE IN `validatePosition` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/JungleSetting.sol: 299, 308, 317 | Acknowledged |

## Description

The `validatePosition` function is responsible for verifying the size of the current position or additional position being opened. However, it incorrectly utilizes the `_size` parameter to represent the total sum of all user positions, rather than the size of the current additional position being added.

```
299         openInterestPerSide[_isLong] + _size <=
300                 (
301                     maxOpenInterestPerSide[_isLong] > 0
302                         ? maxOpenInterestPerSide[_isLong]
303                         : DEFAULT_MAX_OPEN_INTEREST
304                 ),
305             "exceed max open interest per side"
306          );
```

## Recommendation

Adjust the parameter usage to correctly reflect the size of the position being evaluated. Implement thorough testing to validate the correctness of the function's behavior under various scenarios.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team acknowledged the finding.

# JSB-02 INCORRECT HANDLING OF DECREASED POSITION IN `decreaseOpenInterest` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/JungleSetting.sol: 473 | Resolved |

## Description

The `decreaseOpenInterest()` function is utilized to decrease multiple contract variables `openInterestPerAsset`, `openInterestPerSide`, `openInterestAssetPerSide` tracking user position quantity information when a user's position decreases. However, when the position to be decreased exceeds a certain variable's value, instead of assigning it to 0, the variable is decreased by 0. These variables impact the calculation result of `fundingfee`.

```
473     openInterestPerAsset[_token] -= 0;
```

```
479     openInterestPerSide[_isLong] -= 0;
```

```
485     openInterestAssetPerSide[_token][_isLong] -= 0;
```

## Recommendation

Ensure that when a position decreases, variables are appropriately adjusted, including assigning them to 0 when necessary.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: c47c947e9a4a6e0e9fbfad59c49d65c9663ba83f.

# JUN-02 | UNCHECKED RETURN VALUE IN `isClose` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/Jungle.sol: 119, 366 | Acknowledged |

## Description

The `isClose` function is designed to verify whether the current `indexToken` has been closed. It returns a boolean value indicating the status. However, the return value of this function is not being checked within the code. Consequently, even if the market for the `indexToken` has been closed, users can still proceed to open positions.

## Recommendation

Implement a function to validate the return value of the `isClose` function to ensure that users cannot open positions if the market for the corresponding `indexToken` has been closed.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: Issue acknowledged. I won't make any changes for the current version.

In current version, all indexTokens will be open. But we will fix it in the future version if necessary.

# JUN-03 UNAUTHORIZED CANCELLATION OF KEEPER'S ORDERS IN `cancelLimitOrder` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/Jungle.sol: 310 | Resolved |

## Description

The `cancelLimitOrder()` function, designed to cancel user limit orders, contains a logic flaw that allows unauthorized cancellation of orders belonging to a Keeper. When parameter `_account` with Keeper privileges is provided, anyone can cancel orders associated with that Keeper.

```
310     if(keepers[_account]) account = _account;
```

## Proof of Concept

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface Jungle {
    function cancleLimitOrder(address _account,
        uint256 _index, bool _isIncrease
    ) external;
}

contract CancleTheOrder {
    Jungle public jungle;


    ... other logic set keepers

    function cancle() external {

        ... other logic

        jungle.cancleLimitOrder(keeper, _index, _isIncrease);
        bytes32 key = Positions.getRequestKey(keeper, _index);
        IncreasePositionRequest memory request =
jungle.increasePositionRequests(key);
        require(request.indexToken == bytes32(0), "request not exists");

        ... other logic
    }
}
```

## Recommendation

We recommend checking `account` in the if condition instead of `_account`.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: 3b9736bd8043b7f0dd53a63a1d39c9b152ff0cba.

# LPA-03 INCONSISTENT ACCESS CONTROL LEADING TO FUNCTION UNAVAILABILITY

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency, Access Control | Medium | core/LPMarket.sol: 838, 1098 | Resolved |

## Description

A smart contract includes a function with access control mechanisms that restrict its usage to only accounts with specific roles. However, this function is being invoked by another, higher-level function that lacks similar access controls. As a result, unauthorized users attempting to call the higher-level function will fail due to the access control enforced on the underlying function, leading to a malfunction in the intended use of the contract.

## Recommendation

Review and revise the access control logic to ensure that any function calling the restricted function also has appropriate access controls in place. This will prevent unauthorized access at any level of the function call hierarchy.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: f58dabdb0a461e8b5cf30074dc9d553b03f82d2c.

# LPA-07 THE VARIABLE `poolStates` IS NOT UPDATED WHEN PERFORMING LIQUIDATION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | core/LPMarket.sol: 1073 | Acknowledged |

## Description

In the commit `ef6f80413c5c1260b33c0ebe8017314c6abb2c69` : The `liquidatePosition()` function allows the `Manager` to clear positions in the proxy pool. However, there is a logic problem at line 1073, where if the liquidation status does not equal 3, the `lpPositions[_posId]` variable is cleared without updating `poolStates[_poolId]` . This inconsistency can result in inaccurate profit calculations for the proxy pool.

```
1073            _updatePackedPositionX(
1074                _poolId,
1075                !lpPosition.isLong,
1076                false,
1077                lpPosition.averagePrice,
1078                lpPosition.contractSize,
1079                lpPosition.collateral
1080            );
1081
1082            emit ClosePosition(
1083                indexToken,
1084                _poolId,
1085                _posId,
1086                _calcuPositionSize(lpPosition.contractSize, entryPrice),
1087                entryPrice,
1088                data,
1089                lpPosition.isLong,
1090                false
1091            );
1092            delete lpPositions[_posId];
```

## Recommendation

Review the logic in the `liquidatePosition()` function to ensure consistency in variable updates. Make sure that all related variables, such as `poolStates[_poolId]` , are updated during the liquidation process to prevent errors in profit calculations.

## Alleviation

**[Jungle Exchange Team, 05/13/2024]**: Issue acknowledged. I won't make any changes for the current version. In this case liquidation status does not equal 3, we don't need to update poolStates[_poolId] for there is no extra collateral needed to give

back to poolStates.

# LPM-01 | LACK OF ACCESS CONTROL

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Access Control | Medium | core/LPManager.sol: 94 | Resolved |

## Description

The function `createMarket()` can be called by anyone as it has no access restriction. This enables anyone to call this and create the market.

## Recommendation

It's important to implement proper access control mechanisms to protect against such vulnerabilities, such as using a modifier to control who can call this function.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: a22a909dc094ee10af0e9c88b9f7a0322771657d.

# PFB-01 MISSING VALIDATION ON THE RETURN VALUE OF ORACLE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Medium | core/PriceFeed.sol: 38~50 | Acknowledged |

## Description

The function misses proper validations and checks when utilizing price data provided by oracles to ensure its accuracy and timeliness. In the absence of such safeguards, smart contracts may utilize incorrect or outdated price information, which can create economic vulnerabilities that malicious actors could exploit to manipulate or attack the protocol, potentially leading to financial losses.

## Recommendation

To address these vulnerabilities, it is recommended to add additional security checks within the smart contract to ensure that the price data being processed is within reasonable limits and reflects the latest market conditions.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team acknowledged the finding.

# CON-01 | POSSIBLE INTEGER OVERFLOW AND DEAD LOOP

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/JungleSetting.sol: 144; core/LPManager.sol: 364, 373, 379; core/LPMarket.sol: 364~365, 652~653; core/MarketTick.sol: 304; periphery/JungleReader.sol: 44; periphery/LPReader.sol: 188 | Resolved |

## Description

The max value for `uint8` is 255. The `_indexTokens.length` is of type `uint256` and may be bigger than 255. Thus `i++` may silently overflow and it becomes a dead loop.

## Recommendation

Ensure that loop variables are appropriately typed to avoid potential overflow issues.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: 98b4dc6d640891246dc23eeb68a40e27a820c50a.

# COR-03 | QUESTION RELATED TO BID-ASK SPREAD

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/LPManager.sol: 190~195; core/LPMarket.sol: 133, 234~240; core/MarketTick.sol: 113~164, 200~206, 234~240, 252~257, 285~291 | Resolved |

## Description

The `editPool()` function in the smart contract is designed to update the `bidSpread` and `askSpread` parameters of a pool's strategy. These changes subsequently affect the values of `tickBid` and `tickAsk` through the `updateLiquidity` function. The state of `tickPoolBitmap[tickBid]` and `tickPoolBitmap[tickAsk]` in the `MarketTick` contract is altered as a result of the updated tick values. However, the function fails to address the state change for the previous `tickBid` and `tickAsk` values. This may cause inaccurate calculation in searching for the next tick.

## Recommendation

We recommend the team to review the design and provide more illustrations on it.

## Alleviation

**[Jungle Exchange Team, 04/26/2024]**: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/jungle-official/jungle-synthetics/commit/ef6f80413c5c1260b33c0ebe8017314c6abb2c69

# COR-05 CONTRADICTORY IMPLEMENTATION IN MATCHING MECHANISM

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | core/LPMarket.sol: 649~656; core/MarketTick.sol: 147~148, 227, 278 | Acknowledged |

## Description

The white paper specifies that the order matching mechanism should prioritize newer liquidity pools to incentivize the creation of new MM pools. However, the actual implementation deviates from this intended behavior. The variable `poolCreateTime` is only utilized in reward calculation rather than during the order matching process. The function `_matchPools` is designed to match orders with pools, but it prioritize older pools.

```
1             for (
2                 uint8 i = 0;
3                 i < poolStateIds.length && _matchParam.sizeRemaining != 0;
4                 i++
5             ) {
6                 bytes32 poolId = poolStateIds[i];
7                 if (poolId == _matchParam.excludedPoolId) continue;
8                 if (
9                     !_checkPriceRangeAndOrder(
10                        poolId,
11                        _stepParam.price,
12                        _openLong
13                    )
14                ) continue;
15
16                _matchOnePool(poolId, _matchParam, _stepParam, _openLong);
17            }
```

The above implementation is based on traversals on the `tickPools`, and this giving precedence to the pools created earlier or those that have older last-changed timestamps.

## Recommendation

We recommend the team reviewing the implementation and providing illustrations on related design.

## Alleviation

**[Jungle Exchange Team, 04/23/2024]**: Issue acknowledged. I won't make any changes for the current version.

The actual implementation is the final version and we should change the white paper.

# GLOBAL-01 MISSING IMPLEMENTATION OF INDIVIDUAL POOL PRIORITY

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | | Acknowledged |

## Description

The white paper outlines a specific order-matching process where individual liquidity pools are prioritized over a global liquidity pool. This mechanism is crucial for ensuring that trades are executed in a manner that reflects the intended design and encourages the creation of individual MM pools. Upon reviewing the smart contract code, the terms "global pool" and "individual pools" are missing. Consequently, the contract's order-matching function does not adhere to the described priority system, potentially leading to an execution of trades that do not align with the white paper's specifications.

## Recommendation

It is recommended to update the smart contract to incorporate the individual pool priority mechanism as described in the white paper. This involves introducing the necessary variables and logic to distinguish between individual pools and the global pool. Additionally, the order-matching function should be modified to check and process orders based on the intended priority, ensuring that individual pool orders are matched before any orders from the global pool.

## Alleviation

**[Jungle Exchange Team, 04/23/2024]**: Issue acknowledged. I won't make any changes for the current version.

We abandon the terms "global pool" and "individual pools" and we will change the description in the white paper.

# JUN-04 | THE SURPLUS NATIVE TOKENS ARE NOT RETURNED

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | core/Jungle.sol: 83~115, 338~362 | Acknowledged |

## Description

In the `payable` function, there's a validation to ensure `msg.value` meets the minimum required amount of native tokens. However, it's important to note that if `msg.value` exceeds this amount, the function currently lacks a mechanism to refund the excess. This oversight could lead to unintentional loss of funds for the caller, as any surplus in `msg.value` is not returned. Implementing a refund logic for the excess amount is crucial to prevent the potential loss of caller funds.

## Recommendation

We recommend adding logic for refunding surplus or modifying the validation process to enforce that the amount of native tokens paid by the caller exactly matches the required amount.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: Issue acknowledged. I won't make any changes for the current version.

The msg.value is used to execute limit order as the gas fee by keeper when target price match. So it's hard to pay exactly native token by the caller.

msg.value is evaluated by the current gas fee. So it may be much or less than the gas fee deserved.

According to this consideration, we don't return the native tokens if they surplus.

# JUN-05 | POTENTIAL ORDERS FILLED AT WORSE PRICES

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/Jungle.sol: 267 | Acknowledged |

## Description

The `executeLimitOrder` function is intended to be called by the Keeper to execute user-submitted limit orders. However, due to the sequential execution of orders by the Keeper, if multiple users submit orders with similar target prices, the first order will be executed at the current market's more favorable price, while subsequent orders will be executed at less favorable prices.

```
285     uint256 realPrice = _marketOrderIncrease(request);
```

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

## Alleviation

**[Jungle Exchange Team, 04/23/2024]**: Issue acknowledged. I won't make any changes for the current version.

Maybe the subsequent orders will be executed at more favorable prices than the first executed order.

But whatever we should execute the limit order if the acceptPrice is reasonable..

# LPA-06 POTENTIAL LOGICAL INCONSISTENCIES `liquidatePosition` AND `adl`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/LPMarket.sol: 967 | Acknowledged |

## Description

In the commit `ef6f80413c5c1260b33c0ebe8017314c6abb2c69` : The `liquidatePosition()` function is called by the `Manager` to liquidate proxy pool positions. It uses the proxy pool's price rather than an oracle price. Since these prices can differ, this may result in discrepancies when calculating the proxy pool's profits. This is inconsistent with the logic of the function `adl()` .

```
967     function liquidatePosition(
968         bytes32 _poolId,
969         bytes32 _posId
970     ) external onlyManager {
971         LPPosition memory lpPosition = lpPositions[_posId];
972         (
973             uint256 oraclePrice,
974             uint256 liquidationState,
975             uint256 data
976         ) = validateLiquidation(_poolId, _posId, false);
977         MatchedLiquidity memory mLiquidity = matchLiquidityIndexToken(
978             oraclePrice,
979             lpPosition.contractSize,
980             lpPosition.isLong,
981             false,
982             false,
983             _poolId
984         );
985 
986         _liquidatePosition(
987             _poolId,
988             _posId,
989             liquidationState,
990             data,
991             mLiquidity.averagePrice,
992             lpPosition
993         );
994     }
```

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

## Alleviation

**[Jungle Exchange Team, 05/13/2024]**: Issue acknowledged. I won't make any changes for the current version. We think it does not matter in this case for ADL.

# LPA-08 | POTENTIAL AVERAGE PRICE IS NOT CORRECT

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/LPMarket.sol: 769 | Resolved |

## Description

The `_createLPPosition()` function is used to create new positions in a smart contract. However, when an existing position is found, it merges with the new one without updating the average price. This logic flaw could lead to incorrect calculations and inconsistent state within the contract.

```
769         if (lpPositions[posId].poolId != bytes32(0)) {
770             lpPositions[posId].contractSize += _openableSize;
771             lpPositions[posId].collateral += collateral;
772         }
```

## Recommendation

Review the logic for merging positions in the `_createLPPosition()` function and ensure that all relevant values, including the average price, are updated properly during the merge.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: The team heeded the advice and resolved the issue in commit: ef6f80413c5c1260b33c0ebe8017314c6abb2c69.

# LPM-02 POTENTIAL ALLOW UNAUTHORIZED WITHDRAWALS IN `claimReward` FUNCTION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/LPManager.sol: 297 | Acknowledged |

## Description

The `claimReward()` function can be called by anyone to retrieve rewards, which are withdrawn from the vault contract. As the vault contract's tokens primarily consist of user-added liquidity, withdrawals may result in taking assets belonging to other users.

```
297     function claimReward(bytes32 _indexToken, bytes32 _poolId,
298         uint256 _amount
299     ) external {
300         address lpMarket = lpMarkets[_indexToken];
301         require(lpMarket != address(0), "Invalid Pool.");
302         ILPMarket(lpMarket).claimReward(msg.sender, _poolId, _amount);
303         vault.takeJUSDOut(msg.sender, _amount);
304     }
```

## Recommendation

Restrict access to the claimReward function to authorized users only to prevent unauthorized withdrawals of assets.

## Alleviation

**[Jungle Exchange Team, 04/07/2024]**: Issue acknowledged. I won't make any changes for the current version.

# LPM-03 | INCORRECT BOOLEAN EXPRESSION

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | core/LPManager.sol: 214 | Acknowledged |

## Description

The boolean expression `strategy.bidSpread % _tickSize == 0 || strategy.askSpread % _tickSize == 0` means that either the `bidSpread` or the `askSpread` should be divisible by `_tickSize`. Based on the error message, the implementation should require both `bidSpread` and the `askSpread` to be divisible by `_tickSize`.

## Recommendation

We recommend the team changing `||` to `&&`.

## Alleviation

**[Jungle Exchange, 04/23/2024]**: Issue acknowledged. I won't make any changes for the current version.

`tickSize` is the minimum price spread. Price spread must be multiple of tickSize. It is illegal If tickSize is 10 and bidSpread is 8. It is legal if tickSize is 10 and bidSpread is 20.

# MTB-01 | MISSING SERVICE FEE HANDLING MECHANISM

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | core/MarketTick.sol: 972~978 | Resolved |

## Description

When liquidity providers remove liquidity, the platform will charge service fees. According to the white paper, "this portion of the fees is exclusively aimed at LPs and will be partially reimbursed to liquidity pool creators based on the platform's operational conditions. The remaining portion will serve as platform revenue to support long-term development." However, the project lacks of related mechanism of service fees and those fees are currently being fully reimbursed to the pool.

## Recommendation

We recommend the team to implement as white paper instructed to prevent the accumulation of fees in the vault contract.

## Alleviation

**[Jungle Exchange Team, 04/26/2024]**: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/jungle-official/jungle-synthetics/commit/73de5a0845dfa3f58c94a448ed73152fb8559450

# PFB-02 | THIRD-PARTY DEPENDENCY USAGE

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Design Issue | Minor | core/PriceFeed.sol: 32 | Acknowledged |

## Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
44      (, int256 tokenPrice,,,) = aggregators[_tokenName].latestRoundData();
```

- The contract `PriceFeed` interacts with third party contract with `AggregatorV3Interface`.

## Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

## Alleviation

**[Jungle Exchange Team, 04/23/2024]**: We will monitor the statuses of third parties.

# APPENDIX | JUNGLE EXCHANGE

## Finding Categories

| Categories | Description |
| --- | --- |
| Access Control | Access Control findings are about security vulnerabilities that make protected assets unsafe. |
| Inconsistency | Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification. |
| Volatile Code | Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities. |
| Logical Issue | Logical Issue findings indicate general implementation issues related to the program logic. |
| Centralization | Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code. |
| Design Issue | Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories. |

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.