METATRUST

Draft

Security Assessment for

124-2022-06-notional-coop (1New-FLP) (1Positive-SP)

July 23, 2023

Executive Summary

Overview	
Project Name	124-2022-06-notional-coop (1New-FLP) (1Positive-SP)
Codebase URL	https://github.com/code-423n4/2022-06-notional-coop
Scan Engine	AI Analyzer
Scan Time	2023/07/23 17:45:05
Commit Id	6f8c325

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.

Total	
Critical Issues	0
High risk Issues	13
Medium risk Issues	0
Low risk Issues	0
Informational Issues	0

Critical Issues	0%	**0**
High risk Issues	100%	**13**
Medium risk Issues	0%	**0**
Low risk Issues	0%	**0**
Informational Issues	0%	**0**

Summary of Findings

MetaScan security assessment was performed on **July 23, 2023 17:45:05** on project **124-2022-06-notional-coop (1New-FLP) (1Positive-SP)** with the repository **https://github.com/code-423n4/2022-06-notional-coop** on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **AI Analyzer**. There are in total **13** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **13** high risk vulnerabilities, **0** medium risk vulnerabilities, **0** low risk vulnerabilities, **0** informational issues.

ID	Description	Severity
MSA-001	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-002	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-003	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-004	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-005	MWE-206: No Slippage Limit Check	High risk
MSA-006	MWE-206: No Slippage Limit Check	High risk
MSA-007	MWE-206: No Slippage Limit Check	High risk
MSA-008	MWE-206: No Slippage Limit Check	High risk
MSA-009	MWE-206: No Slippage Limit Check	High risk
MSA-010	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-011	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-012	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-013	MWE-200: Insecure LP Token Value Calculation	High risk

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (13)

1. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/deprecated/NAVIssuanceModule.sol #942-967

```
    function _getSetTokenMintQuantity(
        ISetToken _setToken,
        address _reserveAsset,
        uint256 _netReserveFlows,            // Value of reserve asset net of fees
        uint256 _setTotalSupply
    )
        internal
        view
        returns (uint256)
    {
        uint256 premiumPercentage = _getIssuePremium(_setToken, _reserveAsset, _netReserveFlows);
        uint256 premiumValue = _netReserveFlows.preciseMul(premiumPercentage);

        // Get valuation of the SetToken with the quote asset as the reserve asset. Returns value in pr
        // Reverts if price is not found
        uint256 setTokenValuation = controller.getSetValuer().calculateSetTokenValuation(_setToken, _re

        // Get reserve asset decimals
        uint256 reserveAssetDecimals = ERC20(_reserveAsset).decimals();
        uint256 normalizedTotalReserveQuantityNetFees = _netReserveFlows.preciseDiv(10 ** reserveAssetD
        uint256 normalizedTotalReserveQuantityNetFeesAndPremium = _netReserveFlows.sub(premiumValue).pr

        // Calculate SetTokens to mint to issuer
        uint256 denominator = _setTotalSupply.preciseMul(setTokenValuation).add(normalizedTotalReserveQ
        return normalizedTotalReserveQuantityNetFeesAndPremium.preciseMul(_setTotalSupply).preciseDiv(c
    }
```

index-coop-notional-trade-module/contracts/protocol/modules/v1/CustomOracleNAVIssuanceModule.sol #679-714

```
    function _createIssuanceInfo(
        ISetToken _setToken,
        address _reserveAsset,
        uint256 _reserveAssetQuantity
    )
        internal
        view
        returns (ActionInfo memory)
    {
        ActionInfo memory issueInfo;

        issueInfo.previousSetTokenSupply = _setToken.totalSupply();

        issueInfo.preFeeReserveQuantity = _reserveAssetQuantity;

        (issueInfo.protocolFees, issueInfo.managerFee, issueInfo.netFlowQuantity) = _getFees(
            _setToken,
            issueInfo.preFeeReserveQuantity,
            PROTOCOL_ISSUE_MANAGER_REVENUE_SHARE_FEE_INDEX,
            PROTOCOL_ISSUE_DIRECT_FEE_INDEX,
            MANAGER_ISSUE_FEE_INDEX
        );

        issueInfo.setTokenQuantity = _getSetTokenMintQuantity(
            _setToken,
            _reserveAsset,
            issueInfo.netFlowQuantity,
            issueInfo.previousSetTokenSupply
        );

        (issueInfo.newSetTokenSupply, issueInfo.newPositionMultiplier) = _getIssuePositionMultiplier(_s

        issueInfo.newReservePositionUnit = _getIssuePositionUnit(_setToken, _reserveAsset, issueInfo);

        return issueInfo;
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

2. MWE-200: Insecure LP Token Value Calculation

High risk

Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/PriceOracle.sol #323-343

```
function _getPriceFromAdapters(
    address _assetOne,
    address _assetTwo
)
    internal
    view
    returns (bool, uint256)
{
    for (uint256 i = 0; i < adapters.length; i++) {
        (
            bool priceFound,
            uint256 price
        ) = IOracleAdapter(adapters[i]).getPrice(_assetOne, _assetTwo);

        if (priceFound) {
            return (priceFound, price);
        }
    }

    return (false, 0);
}
```

index-coop-notional-trade-module/contracts/protocol/PriceOracle.sol #117-139

```
function getPrice(address _assetOne, address _assetTwo) external view returns (uint256) {
    require(
        controller.isSystemContract(msg.sender),
        "PriceOracle.getPrice: Caller must be system contract."
    );

    bool priceFound;
    uint256 price;

    (priceFound, price) = _getDirectOrInversePrice(_assetOne, _assetTwo);

    if (!priceFound) {
        (priceFound, price) = _getPriceFromMasterQuote(_assetOne, _assetTwo);
    }

    if (!priceFound) {
        (priceFound, price) = _getPriceFromAdapters(_assetOne, _assetTwo);
    }

    require(priceFound, "PriceOracle.getPrice: Price not found.");

    return price;
}
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

3. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/SetValuer.sol #83-111

```
    function calculateSetTokenValuation(ISetToken _setToken, address _quoteAsset) external view returns
        IPriceOracle priceOracle = controller.getPriceOracle();
        address masterQuoteAsset = priceOracle.masterQuoteAsset();
        address[] memory components = _setToken.getComponents();
        int256 valuation;

        for (uint256 i = 0; i < components.length; i++) {
            address component = components[i];
            // Get component price from price oracle. If price does not exist, revert.
            uint256 componentPrice = priceOracle.getPrice(component, masterQuoteAsset);

            int256 aggregateUnits = _setToken.getTotalComponentRealUnits(component);

            // Normalize each position unit to preciseUnits 1e18 and cast to signed int
            uint256 unitDecimals = ERC20(component).decimals();
            uint256 baseUnits = 10 ** unitDecimals;
            int256 normalizedUnits = aggregateUnits.preciseDiv(baseUnits.toInt256());

            // Calculate valuation of the component. Debt positions are effectively subtracted
            valuation = normalizedUnits.preciseMul(componentPrice.toInt256()).add(valuation);
        }

        if (masterQuoteAsset != _quoteAsset) {
            uint256 quoteToMaster = priceOracle.getPrice(_quoteAsset, masterQuoteAsset);
            valuation = valuation.preciseDiv(quoteToMaster.toInt256());
        }

        return valuation.toUint256();
    }
```

index-coop-notional-trade-module/deprecated/NAVIssuanceModule.sol #942-967

```
    function _getSetTokenMintQuantity(
        ISetToken _setToken,
        address _reserveAsset,
        uint256 _netReserveFlows,            // Value of reserve asset net of fees
        uint256 _setTotalSupply
    )
        internal
        view
        returns (uint256)
    {
        uint256 premiumPercentage = _getIssuePremium(_setToken, _reserveAsset, _netReserveFlows);
        uint256 premiumValue = _netReserveFlows.preciseMul(premiumPercentage);

        // Get valuation of the SetToken with the quote asset as the reserve asset. Returns value in p
        // Reverts if price is not found
        uint256 setTokenValuation = controller.getSetValuer().calculateSetTokenValuation(_setToken, _re

        // Get reserve asset decimals
        uint256 reserveAssetDecimals = ERC20(_reserveAsset).decimals();
        uint256 normalizedTotalReserveQuantityNetFees = _netReserveFlows.preciseDiv(10 ** reserveAsset
        uint256 normalizedTotalReserveQuantityNetFeesAndPremium = _netReserveFlows.sub(premiumValue).p

        // Calculate SetTokens to mint to issuer
        uint256 denominator = _setTotalSupply.preciseMul(setTokenValuation).add(normalizedTotalReserve
        return normalizedTotalReserveQuantityNetFeesAndPremium.preciseMul(_setTotalSupply).preciseDiv(
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

4. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/CompoundLeverageModule.sol #803-843

```
    function _executeTrade(
        ActionInfo memory _actionInfo,
        IERC20 _sendToken,
        IERC20 _receiveToken,
        bytes memory _data
    )
        internal
        returns (uint256)
    {
         ISetToken setToken = _actionInfo.setToken;
         uint256 notionalSendQuantity = _actionInfo.notionalSendQuantity;

         setToken.invokeApprove(
            address(_sendToken),
            _actionInfo.exchangeAdapter.getSpender(),
            notionalSendQuantity
        );

        (
            address targetExchange,
            uint256 callValue,
            bytes memory methodData
        ) = _actionInfo.exchangeAdapter.getTradeCalldata(
            address(_sendToken),
            address(_receiveToken),
            address(setToken),
            notionalSendQuantity,
            _actionInfo.minNotionalReceiveQuantity,
            _data
        );

        setToken.invoke(targetExchange, callValue, methodData);

        uint256 receiveTokenQuantity = _receiveToken.balanceOf(address(setToken)).sub(_actionInfo.preTr
        require(
            receiveTokenQuantity >= _actionInfo.minNotionalReceiveQuantity,
            "Slippage too high"
        );

        return receiveTokenQuantity;
    }
```

index-coop-notional-trade-module/contracts/protocol/modules/v1/CompoundLeverageModule.sol #322-374

```
    function deleverToZeroBorrowBalance(
        ISetToken _setToken,
        IERC20 _collateralAsset,
        IERC20 _repayAsset,
        uint256 _redeemQuantity,
        string memory _tradeAdapterName,
        bytes memory _tradeData
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        uint256 notionalRedeemQuantity = _redeemQuantity.preciseMul(_setToken.totalSupply());

        require(borrowCTokenEnabled[_setToken][underlyingToCToken[_repayAsset]], "Borrow not enabled");
        uint256 notionalRepayQuantity = underlyingToCToken[_repayAsset].borrowBalanceCurrent(address(_s

        ActionInfo memory deleverInfo = _createAndValidateActionInfoNotional(
            _setToken,
            _collateralAsset,
            _repayAsset,
            notionalRedeemQuantity,
            notionalRepayQuantity,
            _tradeAdapterName,
            false
        );

        _redeemUnderlying(deleverInfo.setToken, deleverInfo.collateralCTokenAsset, deleverInfo.notional

        _executeTrade(deleverInfo, _collateralAsset, _repayAsset, _tradeData);

        // We use notionalRepayQuantity vs. Compound's max value uint256(-1) to handle WETH properly
        _repayBorrow(deleverInfo.setToken, deleverInfo.borrowCTokenAsset, _repayAsset, notionalRepayQua

        // Update default position first to save gas on editing borrow position
        _setToken.calculateAndEditDefaultPosition(
            address(_repayAsset),
            deleverInfo.setTotalSupply,
            deleverInfo.preTradeReceiveTokenBalance
        );

        _updateLeverPositions(deleverInfo, _repayAsset);

        emit LeverageDecreased(
            _setToken,
            _collateralAsset,
            _repayAsset,
            deleverInfo.exchangeAdapter,
            deleverInfo.notionalSendQuantity,
            notionalRepayQuantity,
            0 // No protocol fee
        );
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

5. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #206-244

```
    function removeLiquidity(
        ISetToken _setToken,
        string memory _ammName,
        address _ammPool,
        uint256 _poolTokenPositionUnits,
        address[] calldata _components,
        uint256[] calldata _minComponentUnitsReceived
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        ActionInfo memory actionInfo = _getActionInfo(
            _setToken,
            _ammName,
            _ammPool,
            _components,
            _minComponentUnitsReceived,
            _poolTokenPositionUnits
        );

        _validateRemoveLiquidity(actionInfo);

        _executeRemoveLiquidity(actionInfo);

        _validateMinimumUnderlyingReceived(actionInfo);

        int256 liquidityTokenDelta = _updateLiquidityTokenPositions(actionInfo);

        int256[] memory componentsDelta = _updateComponentPositions(actionInfo);

        emit LiquidityRemoved(
            _setToken,
            _ammPool,
            liquidityTokenDelta,
            _components,
            componentsDelta
        );
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

6. MWE-206: No Slippage Limit Check

High risk

Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #462-480

```
    function _executeRemoveLiquidity(ActionInfo memory _actionInfo) internal {
        (
            address targetAmm, uint256 callValue, bytes memory methodData
        ) = _actionInfo.ammAdapter.getRemoveLiquidityCalldata(
            address(_actionInfo.setToken),
            _actionInfo.liquidityToken,
            _actionInfo.components,
            _actionInfo.totalNotionalComponents,
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invokeApprove(
            _actionInfo.liquidityToken,
            _actionInfo.ammAdapter.getSpenderAddress(_actionInfo.liquidityToken),
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invoke(targetAmm, callValue, methodData);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

7. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #462-480

```
    function _executeRemoveLiquidity(ActionInfo memory _actionInfo) internal {
        (
            address targetAmm, uint256 callValue, bytes memory methodData
        ) = _actionInfo.ammAdapter.getRemoveLiquidityCalldata(
            address(_actionInfo.setToken),
            _actionInfo.liquidityToken,
            _actionInfo.components,
            _actionInfo.totalNotionalComponents,
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invokeApprove(
            _actionInfo.liquidityToken,
            _actionInfo.ammAdapter.getSpenderAddress(_actionInfo.liquidityToken),
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invoke(targetAmm, callValue, methodData);
    }
```

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #206-244

```
    function removeLiquidity(
        ISetToken _setToken,
        string memory _ammName,
        address _ammPool,
        uint256 _poolTokenPositionUnits,
        address[] calldata _components,
        uint256[] calldata _minComponentUnitsReceived
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        ActionInfo memory actionInfo = _getActionInfo(
            _setToken,
            _ammName,
            _ammPool,
            _components,
            _minComponentUnitsReceived,
            _poolTokenPositionUnits
        );

        _validateRemoveLiquidity(actionInfo);

        _executeRemoveLiquidity(actionInfo);

        _validateMinimumUnderlyingReceived(actionInfo);

        int256 liquidityTokenDelta = _updateLiquidityTokenPositions(actionInfo);

        int256[] memory componentsDelta = _updateComponentPositions(actionInfo);

        emit LiquidityRemoved(
            _setToken,
            _ammPool,
            liquidityTokenDelta,
            _components,
            componentsDelta
        );
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

8. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #482-500

```
    function _executeRemoveLiquiditySingleAsset(ActionInfo memory _actionInfo) internal {
        (
            address targetAmm, uint256 callValue, bytes memory methodData
        ) = _actionInfo.ammAdapter.getRemoveLiquiditySingleAssetCalldata(
            address(_actionInfo.setToken),
            _actionInfo.liquidityToken,
            _actionInfo.components[0],
            _actionInfo.totalNotionalComponents[0],
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invokeApprove(
            _actionInfo.liquidityToken,
            _actionInfo.ammAdapter.getSpenderAddress(_actionInfo.liquidityToken),
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invoke(targetAmm, callValue, methodData);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

9. MWE-206: No Slippage Limit Check

High risk

Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #482-500

```
    function _executeRemoveLiquiditySingleAsset(ActionInfo memory _actionInfo) internal {
        (
            address targetAmm, uint256 callValue, bytes memory methodData
        ) = _actionInfo.ammAdapter.getRemoveLiquiditySingleAssetCalldata(
            address(_actionInfo.setToken),
            _actionInfo.liquidityToken,
            _actionInfo.components[0],
            _actionInfo.totalNotionalComponents[0],
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invokeApprove(
            _actionInfo.liquidityToken,
            _actionInfo.ammAdapter.getSpenderAddress(_actionInfo.liquidityToken),
            _actionInfo.liquidityQuantity
        );

        _actionInfo.setToken.invoke(targetAmm, callValue, methodData);
    }
```

index-coop-notional-trade-module/contracts/protocol/modules/v1/AmmModule.sol #258-296

```
    function removeLiquiditySingleAsset(
        ISetToken _setToken,
        string memory _ammName,
        address _ammPool,
        uint256 _poolTokenPositionUnits,
        address _component,
        uint256 _minComponentUnitsReceived
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        ActionInfo memory actionInfo = _getActionInfoSingleAsset(
            _setToken,
            _ammName,
            _ammPool,
            _component,
            _minComponentUnitsReceived,
            _poolTokenPositionUnits
        );

        _validateRemoveLiquidity(actionInfo);

        _executeRemoveLiquiditySingleAsset(actionInfo);

        _validateMinimumUnderlyingReceived(actionInfo);

        int256 liquidityTokenDelta = _updateLiquidityTokenPositions(actionInfo);

        int256[] memory componentsDelta = _updateComponentPositions(actionInfo);

        emit LiquidityRemoved(
            _setToken,
            _ammPool,
            liquidityTokenDelta,
            actionInfo.components,
            componentsDelta
        );
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

10. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/CustomOracleNAVIssuanceModule.sol #943-969

```
    function _getSetTokenMintQuantity(
        ISetToken _setToken,
        address _reserveAsset,
        uint256 _netReserveFlows,            // Value of reserve asset net of fees
        uint256 _setTotalSupply
    )
        internal
        view
        returns (uint256)
    {
        uint256 premiumPercentage = _getIssuePremium(_setToken, _reserveAsset, _netReserveFlows);
        uint256 premiumValue = _netReserveFlows.preciseMul(premiumPercentage);

        // If the set manager provided a custom valuer at initialization time, use it. Otherwise get i
        // Get valuation of the SetToken with the quote asset as the reserve asset. Returns value in p
        // Reverts if price is not found
        uint256 setTokenValuation = _getSetValuer(_setToken).calculateSetTokenValuation(_setToken, _res

        // Get reserve asset decimals
        uint256 reserveAssetDecimals = ERC20(_reserveAsset).decimals();
        uint256 normalizedTotalReserveQuantityNetFees = _netReserveFlows.preciseDiv(10 ** reserveAssetD
        uint256 normalizedTotalReserveQuantityNetFeesAndPremium = _netReserveFlows.sub(premiumValue).p

        // Calculate SetTokens to mint to issuer
        uint256 denominator = _setTotalSupply.preciseMul(setTokenValuation).add(normalizedTotalReserve(
        return normalizedTotalReserveQuantityNetFeesAndPremium.preciseMul(_setTotalSupply).preciseDiv(c
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

11. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/modules/v1/AaveLeverageModule.sol #781-821

```
    function _executeTrade(
        ActionInfo memory _actionInfo,
        IERC20 _sendToken,
        IERC20 _receiveToken,
        bytes memory _data
    )
        internal
        returns (uint256)
    {
        ISetToken setToken = _actionInfo.setToken;
        uint256 notionalSendQuantity = _actionInfo.notionalSendQuantity;

        setToken.invokeApprove(
            address(_sendToken),
            _actionInfo.exchangeAdapter.getSpender(),
            notionalSendQuantity
        );

        (
            address targetExchange,
            uint256 callValue,
            bytes memory methodData
        ) = _actionInfo.exchangeAdapter.getTradeCalldata(
            address(_sendToken),
            address(_receiveToken),
            address(setToken),
            notionalSendQuantity,
            _actionInfo.minNotionalReceiveQuantity,
            _data
        );

        setToken.invoke(targetExchange, callValue, methodData);

        uint256 receiveTokenQuantity = _receiveToken.balanceOf(address(setToken)).sub(_actionInfo.preTr
        require(
            receiveTokenQuantity >= _actionInfo.minNotionalReceiveQuantity,
            "Slippage too high"
        );

        return receiveTokenQuantity;
    }
```

index-coop-notional-trade-module/contracts/protocol/modules/v1/CompoundLeverageModule.sol #322-374

```
    function deleverToZeroBorrowBalance(
        ISetToken _setToken,
        IERC20 _collateralAsset,
        IERC20 _repayAsset,
        uint256 _redeemQuantity,
        string memory _tradeAdapterName,
        bytes memory _tradeData
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        uint256 notionalRedeemQuantity = _redeemQuantity.preciseMul(_setToken.totalSupply());

        require(borrowCTokenEnabled[_setToken][underlyingToCToken[_repayAsset]], "Borrow not enabled");
        uint256 notionalRepayQuantity = underlyingToCToken[_repayAsset].borrowBalanceCurrent(address(_s

        ActionInfo memory deleverInfo = _createAndValidateActionInfoNotional(
            _setToken,
            _collateralAsset,
            _repayAsset,
            notionalRedeemQuantity,
            notionalRepayQuantity,
            _tradeAdapterName,
            false
        );

        _redeemUnderlying(deleverInfo.setToken, deleverInfo.collateralCTokenAsset, deleverInfo.notional

        _executeTrade(deleverInfo, _collateralAsset, _repayAsset, _tradeData);

        // We use notionalRepayQuantity vs. Compound's max value uint256(-1) to handle WETH properly
        _repayBorrow(deleverInfo.setToken, deleverInfo.borrowCTokenAsset, _repayAsset, notionalRepayQua

        // Update default position first to save gas on editing borrow position
        _setToken.calculateAndEditDefaultPosition(
            address(_repayAsset),
            deleverInfo.setTotalSupply,
            deleverInfo.preTradeReceiveTokenBalance
        );

        _updateLeverPositions(deleverInfo, _repayAsset);

        emit LeverageDecreased(
            _setToken,
            _collateralAsset,
            _repayAsset,
            deleverInfo.exchangeAdapter,
            deleverInfo.notionalSendQuantity,
            notionalRepayQuantity,
            0 // No protocol fee
        );
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

12. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/integration/oracles/UniswapPairPriceAdapter.sol #120-149

```
    function getPrice(address _assetOne, address _assetTwo) external view returns (bool, uint256) {
        require(controller.isSystemContract(msg.sender), "Must be system contract");

        bool isAllowedUniswapPoolOne = uniswapPoolsToSettings[_assetOne].isValid;
        bool isAllowedUniswapPoolTwo = uniswapPoolsToSettings[_assetTwo].isValid;

        // If assetOne and assetTwo are both not Uniswap pools, then return false
        if (!isAllowedUniswapPoolOne && !isAllowedUniswapPoolTwo) {
            return (false, 0);
        }

        IPriceOracle priceOracle = controller.getPriceOracle();
        address masterQuoteAsset = priceOracle.masterQuoteAsset();

        uint256 assetOnePriceToMaster;
        if(isAllowedUniswapPoolOne) {
            assetOnePriceToMaster = _getUniswapPrice(priceOracle, _assetOne, masterQuoteAsset);
        } else {
            assetOnePriceToMaster = priceOracle.getPrice(_assetOne, masterQuoteAsset);
        }

        uint256 assetTwoPriceToMaster;
        if(isAllowedUniswapPoolTwo) {
            assetTwoPriceToMaster = _getUniswapPrice(priceOracle, _assetTwo, masterQuoteAsset);
        } else {
            assetTwoPriceToMaster = priceOracle.getPrice(_assetTwo, masterQuoteAsset);
        }

        return (true, assetOnePriceToMaster.preciseDiv(assetTwoPriceToMaster));
    }
```

index-coop-notional-trade-module/contracts/protocol/PriceOracle.sol #323-343

```
    function _getPriceFromAdapters(
        address _assetOne,
        address _assetTwo
    )
        internal
        view
        returns (bool, uint256)
    {
        for (uint256 i = 0; i < adapters.length; i++) {
            (
                bool priceFound,
                uint256 price
            ) = IOracleAdapter(adapters[i]).getPrice(_assetOne, _assetTwo);

            if (priceFound) {
                return (priceFound, price);
            }
        }

        return (false, 0);
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

13. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

index-coop-notional-trade-module/contracts/protocol/integration/oracles/UniswapPairPriceAdapter.sol #186-215

```
    function _getUniswapPrice(
        IPriceOracle _priceOracle,
        address _poolAddress,
        address _masterQuoteAsset
    )
        internal
        view
        returns (uint256)
    {
        PoolSettings memory poolInfo = uniswapPoolsToSettings[_poolAddress];
        IUniswapV2Pair poolToken = IUniswapV2Pair(_poolAddress);

        // Get prices against master quote asset. Note: if prices do not exist, function will revert
        uint256 tokenOnePriceToMaster = _priceOracle.getPrice(poolInfo.tokenOne, _masterQuoteAsset);
        uint256 tokenTwoPriceToMaster = _priceOracle.getPrice(poolInfo.tokenTwo, _masterQuoteAsset);

        // Get reserve amounts
        (
            uint256 tokenOneReserves,
            uint256 tokenTwoReserves
        ) = UniswapV2Library.getReserves(uniswapFactory, poolInfo.tokenOne, poolInfo.tokenTwo);

        uint256 normalizedTokenOneBaseUnit = tokenOneReserves.preciseDiv(poolInfo.tokenOneBaseUnit);
        uint256 normalizedTokenBaseTwoUnits = tokenTwoReserves.preciseDiv(poolInfo.tokenTwoBaseUnit);

        uint256 totalNotionalToMaster = normalizedTokenOneBaseUnit.preciseMul(tokenOnePriceToMaster).ac
        uint256 totalSupply = poolToken.totalSupply();

        return totalNotionalToMaster.preciseDiv(totalSupply);
    }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

Medium risk (0)

No Medium risk vulnerabilities found here

Low risk (0)

No Low risk vulnerabilities found here

Informational (0)

No Informational vulnerabilities found here

Disclaimer

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 MetaTrust’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 MetaTrust 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. MetaTrust’s position is that each company and individual are responsible for their own due diligence and continuous security. MetaTrust’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 MetaTrust 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 124-2022-06-notional-coop (1New-FLP) (1Positive-SP) Security Assessment AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER

APPLICABLE LAW, MetaTrust HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, MetaTrust 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, MetaTrust 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, MetaTrust 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 MetaTrust NOR ANY OF MetaTrust'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. MetaTrust 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 MetaTrust'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 124-2022-06-notional-coop (1New-FLP) (1Positive-SP) Security Assessment MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF MetaTrust 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 MetaTrust 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.