Primex - Protocol

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **June 19th, 2023 - August 25th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	08/25/2023
0.2	Document Edit	08/26/2023
0.3	Document Edit	08/27/2023
0.4	Draft Review	08/28/2023
0.5	Draft Review	08/28/2023
1.0	Remediation Plan	09/15/2023
1.1	Remediation Plan Review	09/18/2023
1.2	Remediation Plan Update	10/18/2023
1.3	Remediation Plan Update Review	10/18/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Primex engaged Halborn to conduct a security assessment on their smart contracts beginning on June 19th, 2023 and ending on August 25th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided about two months for the engagement and assigned a full-time security engineer to verify the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by the Primex team.

1.3 SCOPE

1. IN-SCOPE TREE & COMMIT :

The security assessment was scoped to the following repository of smart contracts:

Commit ID: f809cc0471935013699407dcd9eab63b60cd2e22

REMEDIATION BRANCH/COMMIT IDs :

- 3532cdb3b5f7e59e031804f5d125ac957692cdf9
- ddf6d6a8a296a7cf03dbbd1540cde0f8dcfa490d
- 602ae25439fa7ae86d23069975624a531203d699
- 4aa13cf48667b77d46471a2f02d78727b1287204
- e9d04dedc78151cba0a759b90a3cbf6dd10add7c
- e811a4a1f10a36780459cefb2d1d23090ff09a2f
- 3a038a73cca7ebc2455459378730016f4d78bab2
- 868eb9da1b22c03415b2d244a3bd836d76abf40a
- 53578af0ae4c20291f414baab11df0525f25c635
- 2a2dcc15ec03833d5a7bc42c8c1c29a8f251d583
- 694e59382fb9e378fecef1ef23af23b097c7bc10
- b10edc72fe57411bfeff984f92805d560f3e28eb
- d04314875fefc2e0868bd848de896e58ca287958
- 7ea8a23f760aa5b8e719627792b78a678b30004f

LATEST REPOSITORY/COMMIT ID :

The latest commit checked on the newly created repository is as follows:

- a8a22bcd2a84ad84b6b4644547183ce2d2412d15

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet environment. (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

EXECUTIVE OVERVIEW

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

EXECUTIVE OVERVIEW

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	4	3	10

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) NON-STANDARD ERC20 TOKENS WILL REVERT	Medium (5.6)	SOLVED - 09/15/2023
(HAL-02) RELAX STRICT CONDITIONS ON SWAPS WITHOUT DEBT	Medium (5.6)	SOLVED - 09/15/2023
(HAL-03) batchDecreaseTradersDebt AND decreaseTraderDebt ACCOUNT PERMANENT LOSS IN A DIFFERENT WAY	Medium (5.6)	SOLVED - 09/15/2023
(HAL-04) CHAINLINK latestRoundData MIGHT RETURN INCORRECT RESULTS	Medium (5.6)	RISK ACCEPTED
(HAL-05) IMPLEMENTATION CONTRACTS CAN BE INITIALIZED	Low (3.1)	SOLVED - 09/15/2023
(HAL-06) INCOMPATIBILITY WITH REBASING/DEFLATIONARY/INFLATIONARY TOKENS	Low (3.1)	RISK ACCEPTED
(HAL-07) IF FEE IS PAID ON PMX CONTROL MSG.VALUE IS ZERO	Low (3.1)	SOLVED - 09/15/2023
(HAL-08) CONTROL REVERT IF BUCKET DOES NOT HAVE BALANCE FOR BURNING ALL PTOKEN	Informational (0.0)	SOLVED - 09/15/2023
(HAL-09) EMIT EVENT ON updateIndexes	Informational (0.0)	SOLVED - 09/15/2023
(HAL-10) SOLIDITY VERSION 0.8.20 MAY NOT WORK ON OTHER CHAINS DUE TO PUSH0	Informational (0.0)	SOLVED - 09/15/2023
(HAL-11) USE SHIFT RIGHT/LEFT INSTEAD OF DIVISION MULTIPLICATION IF POSSIBLE	Informational (0.0)	ACKNOWLEDGED
(HAL-12) INITIALIZED VARIABLE TO DEFAULT VALUE	Informational (0.0)	SOLVED - 09/15/2023
(HAL-13) CACHE ARRAY LENGTH OUTSIDE OF LOOP	Informational (0.0)	ACKNOWLEDGED
(HAL-14) USE ++i INSTEAD OF i++ ON FOR LOOPS	Informational (0.0)	ACKNOWLEDGED
(HAL-15) USE CUSTOM ERRORS	Informational (0.0)	SOLVED - 09/15/2023

(HAL-16) USING BOOLS FOR STORAGE INCURS OVERHEAD	Informational (0.0)	ACKNOWLEDGED
(HAL-17) FLOATING PRAGMA	Informational (0.0)	SOLVED - 09/15/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) NON-STANDARD ERC20 TOKENS WILL REVERT - MEDIUM (5.6)

Description:

The library TokenTransfersLibrary.sol contains the function to perform ERC20 tokens transfers in the protocol. However, this library uses the interface of IERC20 from OpenZeppelin which enforces the return value on transfer.

This pattern is not followed by all ERC20 tokens, as for example USDT. If attempting to transfer these tokens, the contract will revert, preventing the transaction to be executed.

Code Location:

TokenTransfersLibrary.sol#L12-L19

Listing 1: TokenTransfersLibrary.sol (Line 17)

```
12 function doTransferFromTo(address token, address from, address to,
 ↳ uint256 amount) public returns (uint256) {
13     uint256 balanceBefore = IERC20(token).balanceOf(to);
14     // The returned value is checked in the assembly code below.
15     // Arbitrary `from` should be checked at a higher level. The
 ↳ library function cannot be called by the user.
16     // slither-disable-next-line unchecked-transfer arbitrary-send
 ↳ -erc20
17     IERC20(token).transferFrom(from, to, amount);
18
19     bool success;
```

TokenTransfersLibrary.sol#L46-L51

Listing 2: TokenTransfersLibrary.sol (Line 49)

```
46 function doTransferOut(address token, address to, uint256 amount)
 ↳ public {
```

```
    // The returned value is checked in the assembly code below.
    // slither-disable-next-line unchecked-transfer
    IERC20(token).transfer(to, amount);

    bool success;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:L/R:N/S:U (5.6)

Recommendation:

Consider using a non-strict interface, as compound does, to transfer ERC20 tokens.

Remediation Plan:

SOLVED: The Primex team solved the issue by using a non-strict interface.

Commit ID: 88d33deeebf9c169d21e333ef871c518b10e0b33

4.2 (HAL-02) RELAX STRICT CONDITIONS ON SWAPS WITHOUT DEBT - MEDIUM (5.6)

Description:

The multiSwap function from the PrimexPricingLibrary.sol contract executes the token swap across the specified DEX and paths for opening and closing orders. However, when executing an order without leverage, the protocol sets the tolerable limit to zero, requiring for the swap to return an exact price given the oracle current prices.

This results in most of sport orders reverting, as the retrieved tokens are normally less than the strict oracle prices exchange.

Code Location:

PrimexPricingLibrary.sol#L353-L359

Listing 3: PrimexPricingLibrary.sol (Line 357)

```
353 if (_needOracleTolerableLimitCheck) {
354     _require(
355         vars.balance >=
356             getOracleAmountsOut(_params.tokenA, _params.tokenB, _params.amountTokenA, _priceOracle).wmul(
357                 WadRayMath.WAD - _maximumOracleTolerableLimit
358             ),
359         Errors.DIFFERENT_PRICE_DEX_AND_ORACLE.selector
360     );
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:L/R:N/S:U (5.6)

Recommendation:

Consider allowing the _maximumOracleTolerableLimit for spot orders or avoid executing this code section as the minimum amount out is also checked and specified by the user.

Remediation Plan:

SOLVED: The Primex team solved the issue by turning off oracle checks for manual closing of spot positions and kept oracle tolerable limit check obligatory for limit orders with non-zero tolerance.

Commit ID: 3532cdb3b5f7e59e031804f5d125ac957692cdf9

4.3 (HAL-03) batchDecreaseTradersDebt AND decreaseTraderDebt ACCOUNT PERMANENT LOSS IN A DIFFERENT WAY - MEDIUM (5.6)

Description:

The functions decreaseTraderDebt and batchDecreaseTradersDebt of the Bucket.sol contract account the permanentLossScaled global state variable differently.

On the decreaseTraderDebt function, the liquidityIndex is first updated and the used the new value to compute the permanentLossScaled value. Whereas on the batchDecreaseTradersDebt function, the current liquidityIndex is used to compute the permanentLossScaled and then the index is updated.

Code Location:

Bucket.sol#L365C1-L383C6

Listing 4: Bucket.sol

```
function decreaseTraderDebt(
    address _trader,
    uint256 _debtToBurn,
    address _receiverOfAmountToReturn,
    uint256 _amountToReturn,
    uint256 _permanentLossAmount
) external override onlyRole(PM_ROLE) {
    // don't need require on isLaunched,
    // because if we can't openPosition in this bucket then we can't closePosition in this bucket
    if (_amountToReturn != 0) {
        TokenTransfersLibrary.doTransferOut(address(borrowedAsset), _receiverOfAmountToReturn, _amountToReturn);
```

```
    }
    _updateIndexes();
    debtToken.burn(_trader, _debtToBurn, variableBorrowIndex);
    _updateRates();
    if (_permanentLossAmount != 0) {
        permanentLossScaled += _permanentLossAmount.rdiv(liquidityIndex);
    }
}
```

Bucket.sol#L388-L407

Listing 5: Bucket.sol

```
function batchDecreaseTradersDebt(
    address[] memory _traders,
    uint256[] memory _debtsToBurn,
    address _receiverOfAmountToReturn,
    uint256 _amountToReturn,
    uint256 _permanentLossAmount,
    uint256 _length
) external override onlyRole(BATCH_MANAGER_ROLE) {
    // don't need require on isLaunched,
    // because if we can't openPosition in this bucket then we can't closePosition in this bucket
    if (_amountToReturn != 0) {
        TokenTransfersLibrary.doTransferOut(address(borrowedAsset), _receiverOfAmountToReturn, _amountToReturn);
    }
    if (_permanentLossAmount != 0) {
        permanentLossScaled += _permanentLossAmount.rdiv(liquidityIndex);
    }
    _updateIndexes();
    debtToken.batchBurn(_traders, _debtsToBurn, variableBorrowIndex, _length);
    _updateRates();
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:L/R:N/S:U (5.6)

Recommendation:

Consider which is the appropriate way to handle the accounting of permanent loss and implement it consistently on both function.

Remediation Plan:

SOLVED: The Primex team solved the issue by implementing suggestions consistently on both of the functions.

Commit IDs:

- 602ae25439fa7ae86d23069975624a531203d699
- 4aa13cf48667b77d46471a2f02d78727b1287204

4.4 (HAL-04) CHAINLINK latestRoundData MIGHT RETURN INCORRECT RESULTS - MEDIUM (5.6)

Description:

The getExchangeRate function from the PriceOracle.sol contract calls the latestRoundData function from ChainLink price feeds. However, there is no check on the return values to validate stale data prices.

This could lead to stale prices according to the ChainLink documentation:

ChainLink Doc. Ref#1
ChainLink Doc. Ref#2

Code Location:

PriceOracle.sol#L73-L97

Listing 6: PriceOracle.sol (Lines 82,83,92)

```
73 function getExchangeRate(address assetA, address assetB) external view override returns (uint256, bool) {
74     address priceFeed = chainLinkPriceFeeds[assetA][assetB];
75     bool isForward = true;
76
77     if (priceFeed == address(0)) {
78         priceFeed = chainLinkPriceFeeds[assetB][assetA];
79         if (priceFeed == address(0)) {
80             (address basePriceFeed, address quotePriceFeed) = getPriceFeedsPair(assetA, assetB);
81
82             (, int256 basePrice, , , ) = AggregatorV3Interface(basePriceFeed).latestRoundData();
83             (, int256 quotePrice, , , ) = AggregatorV3Interface(quotePriceFeed).latestRoundData();
84
85             _require(basePrice > 0 && quotePrice > 0, Errors.ZERO_EXCHANGE_RATE.selector);
```

```
            //the return value will always be 18 decimals if the basePrice and quotePrice have the same decimals
            return (uint256(basePrice).wdiv(uint256(quotePrice)), true);
        }
        isForward = false;
    }

    (, int256 answer, , , ) = AggregatorV3Interface(priceFeed).latestRoundData();
    _require(answer > 0, Errors.ZERO_EXCHANGE_RATE.selector);

    uint256 answerDecimals = AggregatorV3Interface(priceFeed).decimals();
    return ((uint256(answer) * 10 ** (18 - answerDecimals)), isForward);
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:L/R:N/S:U (5.6)

Recommendation:

Consider adding the next code, validate the oracle response.

Listing 7

```
( roundId, rawPrice, , updateTime, answeredInRound ) = AggregatorV3Interface(XXXXX).latestRoundData();
require(rawPrice > 0, "Chainlink price <= 0");
require(updateTime != 0, "Incomplete round");
require(answeredInRound >= roundId, "Stale price");
```

Remediation Plan:

RISK ACCEPTED: The Primex team claims that accepting the risk of outdated oracle data is reasonable because it is logical not to block position liquidation or conditional closing based on such data. By the time the oracle reports the price, these positions should already be closed. Primex has an emergency pause system that allows **EMERGENCY_ADMIN** to pause borrowing funds from credit buckets and forbids opening new positions, etc. Each CL feed has a different heartbeat, and the normal heartbeat value is not stored on-chain, so the behavior of the oracle is monitored off-chain. If an insufficient price update frequency is detected, the corresponding components of the protocol will be paused until the oracle is stabilized. Additionally, the team will provide users in the Primex app with information regarding outdated oracle prices. In future versions, the team is considering implementing reserve oracles for critical situations like these.

4.5 (HAL-05) IMPLEMENTATION CONTRACTS CAN BE INITIALIZED - LOW (3.1)

Description:

The implementation contracts of the protocol that are used by proxies do not disable the initialize function.

An uninitialized contract can be taken over by an attacker, which map impact the proxy or be used for social engineering.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:L/R:N/S:U (3.1)

Recommendation:

Consider adding the _disableInitializers function call on the constructor of each implementation.

Listing 8

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

Remediation Plan:

SOLVED: The Primex team solved the issue by using the _disableInitializers in the constructor.

Commit ID: e9d04dedc78151cba0a759b90a3cbf6dd10add7c

4.6 (HAL-06) INCOMPATIBILITY WITH REBASING/DEFLATIONARY/INFLATIONARY TOKENS - LOW (3.1)

Description:

Functions of Primex Protocol contracts use the TokenTransfersLibrary .sol to handle ERC20 transfers. Although this function returns the difference of balance of the caller before and after the call, this value is not used. As a result, the contract may account to a different value than what was actually transferred to the protocol.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:L/R:N/S:U (3.1)

Recommendation:

Whenever tokens are transferred, the delta of the previous (before transfer) and current (after transfer) token balance should be verified to match the declared token amount.

Remediation Plan:

RISK ACCEPTED: The Primex team accepted the risk of this issue and will not use such tokens in the current version. These ERC20 tokens are planned to be future compatible.

4.7 (HAL-07) IF FEE IS PAID ON PMX CONTROL MSG.VALUE IS ZERO - LOW (3.1)

Description:

The openPosition function of the PositionManager.sol contract allows paying the fee of the execution either on native currency or PMX. However, the payProtocolFee function does not control if the transaction contains msg.value when the function is executed, paying in PMX. This effectively locks the native currency on the PositionManager contract.

Code Location:

PositionManager.sol

Listing 9: PositionManager.sol

```
function openPosition(
    PositionLibrary.OpenPositionParams calldata _params
) external payable override nonReentrant whenNotPaused {
    _notBlackListed();
    (PositionLibrary.Position memory newPosition, PositionLibrary.OpenPositionVars memory vars) = PositionLibrary
        .createPosition(_params, primexDNS, priceOracle);
    PositionLibrary.OpenPositionEventData memory posEventData = _openPosition(newPosition, vars);

    PositionLibrary.Position memory position = positions[positions.length - 1];
    _updateTraderActivity(msg.sender, position.positionAsset, position.positionAmount, position.bucket);
```

PrimexPricingLibrary.sol#L444-L492

Listing 10: PrimexPricingLibrary.sol

```
function payProtocolFee(ProtocolFeeParams memory params) public returns (uint256 protocolFee) {
    ProtocolFeeVars memory vars;
    vars.treasury = params.primexDNS.treasury();
    vars.fromLocked = true;

    if (!params.isByOrder) {
        vars.fromLocked = false;
        params.depositData.protocolFee = getOracleAmountsOut(
            params.depositData.depositedAsset,
            params.feeToken,
            params.depositData.depositedAmount.wmul(params.depositData.leverage).wmul(
                params.feeToken == NATIVE_CURRENCY
                    ? params.primexDNS.protocolRate()
                    : params.primexDNS.protocolRateInPmx()
            ),
            params.priceOracle
        );
        if (params.isSwapFromWallet) {
            if (params.feeToken == NATIVE_CURRENCY) {
                _require(msg.value >= params.depositData.protocolFee, Errors.INSUFFICIENT_DEPOSIT.selector);
                TokenTransfersLibrary.doTransferOutETH(vars.treasury, params.depositData.protocolFee);
                if (msg.value > params.depositData.protocolFee) {
                    TokenTransfersLibrary.doTransferOutETH(
                        params.trader,
                        msg.value - params.depositData.protocolFee
                    );
                }
            } else {
                TokenTransfersLibrary.doTransferFromTo(
                    params.feeToken,
                    params.trader,
                    vars.treasury,
                    params.depositData.protocolFee
                );
            }
            return params.depositData.protocolFee;
        }
    }

```

```
    params.traderBalanceVault.withdrawFrom(
        params.trader,
        vars.treasury,
        params.feeToken,
        params.depositData.protocolFee,
        vars.fromLocked
    );

    return params.depositData.protocolFee;
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:L/R:N/S:U (3.1)

Recommendation:

Consider controlling the msg.value to give back the native currency or revert the transaction if appropriate.

Remediation Plan:

SOLVED: The Primex team solved the issue by adding the check for native currency.

Commit IDs:

- e811a4a1f10a36780459cefb2d1d23090ff09a2f
- 3a038a73cca7ebc2455459378730016f4d78bab2

4.8 (HAL-08) CONTROL REVERT IF BUCKET DOES NOT HAVE BALANCE FOR BURNING ALL PTOKEN - INFORMATIONAL (0.0)

Description:

The Ptoken contract is used to account for the deposit and interest that lenders have on a specific bucket. The interest is not always accrued on time on the bucket, and a lender may attempt to burn its tokens without being enough funds on the bucket.

In this case, the protocol reverts with an error from the ERC20 token contract indicating transferred failed.

Code Location:

Bucket.sol#L314-L350

Listing 11: Bucket.sol

```
314 function withdraw(address _borrowAssetReceiver, uint256 _amount) external override nonReentrant notBlackListed {
315     if (!LMparams.isLaunched) {
316         LMparams.liquidityMiningRewardDistributor.removePoints(name, msg.sender, _amount);
317     } else if (block.timestamp < LMparams.stabilizationPeriodEnd) {
318         _require(
319             _amount <=
320                 pToken.balanceOf(msg.sender) -
321                     LMparams.liquidityMiningRewardDistributor.getLenderAmountInMining(name, msg.sender),
322             Errors.MINING_AMOUNT_WITHDRAW_IS_LOCKED_ON_STABILIZATION_PERIOD.selector
323         );
324     }
325 
```

```
    _updateIndexes();
    uint256 amountToWithdraw = pToken.burn(msg.sender, _amount, liquidityIndex);
    uint256 amountToLender = (WadRayMath.WAD - withdrawalFeeRate).wmul(amountToWithdraw);
    uint256 amountToTreasury = amountToWithdraw - amountToLender;
    if (!LMparams.isLaunched && isInvestEnabled && aaveDeposit > 0) {
        // if liquidity mining failed, take all tokens from aave during first withdraw from bucket
        if (block.timestamp > LMparams.liquidityMiningDeadline) {
            _withdrawAllLiquidityFromAave();
        } else {
            // if liquidity mining is in progress, withdraw needed amount from aave
            address aavePool = dns.aavePool();
            IPool(aavePool).withdraw(address(borrowedAsset), amountToWithdraw, address(this));
            emit WithdrawFromAave(aavePool, amountToWithdraw);
            aaveDeposit -= amountToWithdraw;
        }
    }

    TokenTransfersLibrary.doTransferOut(address(borrowedAsset), dns.treasury(), amountToTreasury);
    emit TopUpTreasury(msg.sender, amountToTreasury);

    TokenTransfersLibrary.doTransferOut(address(borrowedAsset), _borrowAssetReceiver, amountToLender);
    _updateRates();

    emit Withdraw(msg.sender, _borrowAssetReceiver, amountToWithdraw);
}
```

PToken.sol#L169-L198

Listing 12: PToken.sol

```
function burn(address _user, uint256 _amount, uint256 _index) external override onlyBucket returns (uint256) {
    _require(_user != address(0), Errors.ADDRESS_NOT_SUPPORTED.selector);
```

```
    uint256 amountScaled;
    (_amount, amountScaled) = _getValidAmounts(_user, _amount, _index);
    if (address(interestIncreaser) != address(0)) {
        //in this case the _index will be equal to the getNormalizedIncome()
        try interestIncreaser.updateBonus(_user, scaledBalanceOf(_user), address(bucket), _index) {} catch {
            emit Errors.Log(Errors.INTEREST_INCREASER_CALL_FAILED.selector);
        }
    }

    if (address(lenderRewardDistributor) != address(0)) {
        try
            lenderRewardDistributor.updateUserActivity(
                bucket,
                _user,
                scaledBalanceOf(_user),
                (scaledTotalSupply() - amountScaled),
                IActivityRewardDistributor.Role.LENDER
            )
        {} catch {
            emit Errors.Log(Errors.LENDER_REWARD_DISTRIBUTOR_CALL_FAILED.selector);
        }
    }

    _burn(_user, amountScaled);

    emit Burn(_user, _amount);
    return _amount;
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider controlling this scenario and revert with the corresponding error.

Remediation Plan:

SOLVED: The Primex team solved the issue by adding a require statement.

Commit ID: 868eb9da1b22c03415b2d244a3bd836d76abf40a

4.9 (HAL-09) EMIT EVENT ON updateIndexes - INFORMATIONAL (0.0)

Description:

The function _updateIndexes function is a critical operation that updates the liquidityIndex and variableBorrowIndex storage variables. These variables are used to account the amounts that lenders and borrowers should receive/return. But this critical operation does not emit an event to help to monitor the protocol.

Code Location:

Bucket.sol#L576-L590

Listing 13: Bucket .sol

```
576 function _updateIndexes() internal {
577     uint256 cumulatedLiquidityInterest = _calculateLinearInterest(
    ↳ lar, lastUpdatedBlockTimestamp);
578     uint256 newLiquidityIndex = cumulatedLiquidityInterest.rmul(
    ↳ liquidityIndex);
579     _require(newLiquidityIndex <= type(uint128).max, Errors.
    ↳ LIQUIDITY_INDEX_OVERFLOW.selector);
580     liquidityIndex = uint128(newLiquidityIndex);
581
582     uint256 cumulatedVariableBorrowInterest =
    ↳ _calculateCompoundedInterest(bar, lastUpdatedBlockTimestamp);
583     uint256 newVariableBorrowIndex =
    ↳ cumulatedVariableBorrowInterest.rmul(variableBorrowIndex);
584     _require(newVariableBorrowIndex <= type(uint128).max, Errors.
    ↳ BORROW_INDEX_OVERFLOW.selector);
585     uint256 previousVariableBorrowIndex = variableBorrowIndex;
586     variableBorrowIndex = uint128(newVariableBorrowIndex);
587
588     lastUpdatedBlockTimestamp = block.timestamp;
589     _mintToReserve(debtToken.scaledTotalSupply(),
    ↳ previousVariableBorrowIndex, variableBorrowIndex);
590 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider omitting an event to help protocol monitoring.

Remediation Plan:

SOLVED: The Primex team solved the issue by adding another parameter to the event.

Commit ID: 53578af0ae4c20291f414baab11df0525f25c635

4.10 (HAL-10) SOLIDITY VERSION 0.8.20 MAY NOT WORK ON OTHER CHAINS DUE TO PUSH0 - INFORMATIONAL (0.0)

Description:

The introduction of EIP-3855 introduces a breaking change that prevents solidity compiled code to execute on L2 chains. As the protocol expects to work on different chains and contains the floating pragma ˆ0.8.18, it is advised not to use this solidity version.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Avoid using floating pragma that can result to compile with this solc version.

Reference

Remediation Plan:

SOLVED: The Primex team solved the issue by changing the floating pragma with 0.8.18.

Commit ID: 2a2dcc15ec03833d5a7bc42c8c1c29a8f251d583

4.11 (HAL-11) USE SHIFT RIGHT/LEFT INSTEAD OF DIVISION MULTIPLICATION IF POSSIBLE - INFORMATIONAL (0.0)

Description:

While the DIV opcode uses 5 gas, the SHR opcode only uses 3 gas. Furthermore, Solidity's division operation also includes a division-by-0 prevention, which is bypassed using shifting.

Code Location:

FeeExecutor.sol#L311

Listing 14: FeeExecutor.sol

```
307 while (lowest < highest) {
308     if (lowest == highest - 1) break;
309     uint256 mid = (lowest + highest) / 2;
310     uint256 midTimestamp = updatedTimestamps[mid];
311     if (_bonusDeadline < midTimestamp) {
312         highest = mid;
313     } else if (_bonusDeadline > midTimestamp) {
314         lowest = mid;
315     } else {
316         return indexes[midTimestamp];
317     }
318 }
```

Bucket.sol#L714-L715

Listing 15: Bucket.sol

```
701 function _calculateCompoundedInterest(uint256 _bar, uint256
    ↳ _blockTimestamp) internal view returns (uint256) {
702     uint256 exp = block.timestamp - _blockTimestamp;
703 
704     if (exp == 0) {
705         return WadRayMath.RAY;
```

```
    }

    uint256 expMinusOne = exp - 1;
    uint256 expMinusTwo = exp > 2 ? exp - 2 : 0;
    // multiply first to mitigate rounding related issues
    uint256 basePowerTwo = _bar.rmul(_bar) / (SECONDS_PER_YEAR * SECONDS_PER_YEAR);
    uint256 basePowerThree = _bar.rmul(_bar).rmul(_bar) / (SECONDS_PER_YEAR * SECONDS_PER_YEAR * SECONDS_PER_YEAR);

    uint256 secondTerm = (exp * expMinusOne * basePowerTwo) / 2;
    uint256 thirdTerm = (exp * expMinusOne * expMinusTwo * basePowerThree) / 6;

    return WadRayMath.RAY + (_bar * exp) / SECONDS_PER_YEAR + secondTerm + thirdTerm;
}
```

PriceOracle.sol#L36-L39

Listing 16: PriceOracle.sol

```
function increasePairVolatility(
    address _assetA,
    address _assetB,
    uint256 _pairVolatility
) external override onlyRole(EMERGENCY_ADMIN) {
    _require(
        _pairVolatility > pairVolatilities[_assetA][_assetB] && _pairVolatility <= WadRayMath.WAD / 2,
        Errors.PAIRVOLATILITY_IS_NOT_CORRECT.selector
    );
    _setPairVolatility(_assetA, _assetB, _pairVolatility);
}
```

WadRayMath.sol#L17-L34

Listing 17: WadRayMath.sol

```
    function wmul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = add(mul(x, y), WAD / 2) / WAD;
```

```
    }

    //rounds to zero if x*y < WAD / 2
    function rmul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = add(mul(x, y), RAY / 2) / RAY;
    }

    //rounds to zero if x*y < WAD / 2
    function wdiv(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = add(mul(x, WAD), y / 2) / y;
    }

    //rounds to zero if x*y < RAY / 2
    function rdiv(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = add(mul(x, RAY), y / 2) / y;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider using a shift operator instead of diving by a constant to save gas.

Remediation Plan:

ACKNOWLEDGED: The Primex team acknowledged this finding.

4.12 (HAL-12) INITIALIZED VARIABLE TO DEFAULT VALUE - INFORMATIONAL (0.0)

Description:

Initializing variables to the default value executes an extra order that is not required.

Code Location:

FeeExecutor.sol#L294

Listing 18: FeeExecutor.sol

```
uint256 lowest = 0;
```

UniswapInterfaceMulticall.sol#L30

Listing 19: UniswapInterfaceMulticall.sol

```
for (uint256 i = 0; i < calls.length; i++) {
```

LimitOrderLibrary.sol#L469

Listing 20: LimitOrderLibrary.sol

```
for (uint256 i = 0; i < A.length - 1; i++) {
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider avoiding initializing variables to default value.

Remediation Plan:

SOLVED: The Primex team solved the issue by avoiding initializing variables to default value.

Commit ID: 694e59382fb9e378fecef1ef23af23b097c7bc10

4.13 (HAL-13) CACHE ARRAY LENGTH OUTSIDE OF LOOP - INFORMATIONAL (0.0)

Description:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

The identified loops withing the protocol that can be optimized are:

- ActivityRewardDistributor.sol L89, L250
- BatchManager.sol L90, L133, L158, L173, L189, L229, L242, L291
- FeeExecutor.sol L51, L87
- Bucket.sol L76
- DexAdapter.sol L221, L300, L359, L442
- EPMXToken.sol L42, L54
- LimitOrderManager.sol L86, L103, L180, L200, L257, L282
- ReferralProgram.sol L83, L94, L95
- BalancerBotLens.sol L15, L31, L61
- TraderBalanceVault.sol L133
- UniswapInterfaceMulticall.sol L30
- WhiteBlackListBase L33, L45, L51, L65
- BestDexLens.sol L283
- PrimexLens.sol L136, L249, L282, L316, L494
- LimitOrderLibrary L274, L302, L469, L470
- PositionLibrary.sol L480
- PrimexPricingLibrary.sol L172, L139, L145, L176, L188, L193, L268, L272, L324, L333, L341

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider caching the array length before iterating over it.

Remediation Plan:

ACKNOWLEDGED: The Primex team acknowledged the issue.

4.14 (HAL-14) USE ++i INSTEAD OF i++ ON FOR LOOPS - INFORMATIONAL (0.0)

Description:

Using ++i instead of i++ saves 5 gas per loop iteration.

The identified loops withing the protocol that can be optimized are:

- ActivityRewardDistributor.sol L79, L89, L250
- BatchManager.sol L90, L158, L189, L229, L242, L291
- FeeExecutor.sol L51, L87, L131
- Bucket.sol L76, L464
- DebtToken L142, L154
- DexAdapter.sol L221, L300, L359, L442, L462, L478
- EPMXToken.sol L42, L54
- LimitOrderManager.sol L90, L221, L225
- PrimexUpkeep.sol L86, L103, L129, L147, L180, L200, L232, L257, L282, L310
- ReferralProgram.sol L83, L94, L95
- SpotTradingRewardDistributor.sol L160
- BalancerBotLens.sol L15, L31, L61
- TraderBalanceVault.sol L133
- UniswapInterfaceMulticall.sol L30
- WhiteBlackListBase L33, L45, L51, L65
- BestDexLens.sol L283
- PrimexLens.sol L136, L249, L282, L316, L494
- LimitOrderLibrary L274, L302, L469, L470
- PositionLibrary.sol L480
- PrimexPricingLibrary.sol L172, L139, L145, L176, L188, L193, L268, L272, L324, L333, L341

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider replacing i++ to ++i in all indicated for loops.

Remediation Plan:

ACKNOWLEDGED: The Primex team acknowledged the issue.

4.15 (HAL-15) USE CUSTOM ERRORS - INFORMATIONAL (0.0)

Description:

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met). Source Custom Errors in Solidity: Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to provide additional information about failures (e.g., revert(‘‘Insufficient funds.‘‘);), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Code Location:

PrimexPricingLibrary.sol#L553

Listing 21: PrimexPricingLibrary.sol

```
revert("DexAdapter::decodePath: UNKNOWN_DEX_TYPE");
```

WadRayMath.sol#L6-L10

Listing 22: WadRayMath.sol

```
function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
    require((z = x + y) >= x, "ds-math-add-overflow");
}

function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
    require(y == 0 || (z = x * y) / y == x, "ds-math-mul-overflow");
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider replacing strings for custom errors, as done in the rest of the protocol implementation.

Remediation Plan:

SOLVED: The Primex team solved the issue by using custom errors.

Commit ID: b10edc72fe57411bfeff984f92805d560f3e28eb

4.16 (HAL-16) USING BOOLS FOR STORAGE INCURS OVERHEAD - INFORMATIONAL (0.0)

Description:

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past. See source.

Code Location:

Instances (3):

Listing 23

```
File: Bucket/BucketStorage.sol

57:     bool public isInvestEnabled;

```

Listing 24

```
File: EPMXToken.sol

14:     mapping(address => bool) public whitelist;

```

Listing 25

```
File: PMXBonusNFT/PMXBonusNFTStorage.sol

27:     mapping(uint256 => bool) internal isBlocked;

```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider avoiding the usage of boolean types for storage variables.

Remediation Plan:

ACKNOWLEDGED: The Primex team acknowledged the issue.

4.17 (HAL-17) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Primex protocol contract uses the floating pragma ^0.8.18. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

This issue, specifically, aligns with the previous described misbehave on different chains if solidity version 0.8.20 is used.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider locking the pragma version, known bugs for the compiler version. Therefore, it is recommended not to use floating pragma in production.

Remediation Plan:

SOLVED: The Primex team solved the issue by locking pragma to 0.8.18.

Commit ID: 2a2dcc15ec03833d5a7bc42c8c1c29a8f251d583

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

- No major issues found by Slither.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

- No major issues were found by MythX.

THANK YOU FOR CHOOSING

HALBORN