Competitive Security Assessment

uniwhale.co

Feb 28th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	uniwhale.co
Platform & Language	Solidity
Codebase	• https://github.com/uniwhale-io/uniwhale-v1 • audit commit - 58e7ed410d7252f926e92194dc70bafd7049fbd6 • final commit - d9b35fed52122aa06582eeb86409b6cdef68c4b8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

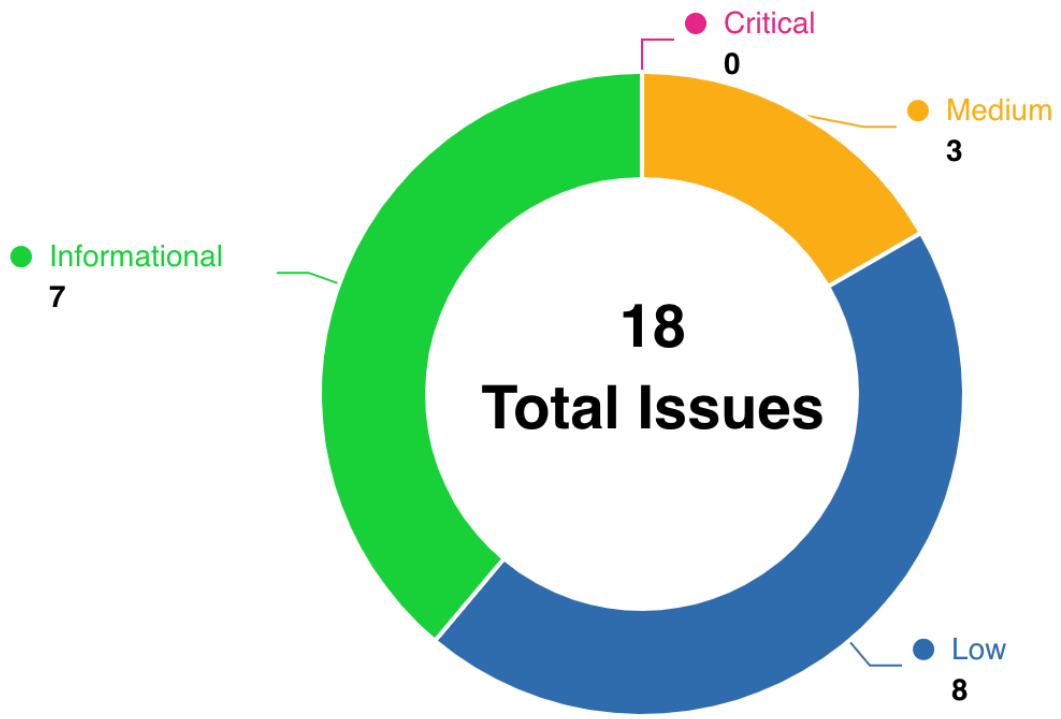
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	3	0	0	2	1	0
Low	8	0	3	4	1	0
Informational	7	0	4	1	1	1

Audit Scope

File	Commit Hash
packages/contracts/core-v1/contracts/TradingCore.sol	58e7ed410d7252f926e92194dc70bafd7049fbd6
packages/contracts/core-v1/contracts/interfaces/AbstractRegistry.sol	58e7ed410d7252f926e92194dc70bafd7049fbd6
packages/contracts/core-v1/contracts/libs/TradingCoreLib.sol	58e7ed410d7252f926e92194dc70bafd7049fbd6
packages/contracts/core-v1/contracts/LiquidityPool.sol	58e7ed410d7252f926e92194dc70bafd7049fbd6
packages/contracts/core-v1/contracts/RegistryCore.sol	58e7ed410d7252f926e92194dc70bafd7049fbd6

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
UNW-1	APPROVED_ROLE is able to remove margin from user's order in TradingCore	Privilege Related	Medium	Mitigated	0xxm
UNW-2	Access control on view function is unnecessary	Gas Optimization	Informational	Acknowledged	0xxm
UNW-3	Centralized risk	Privilege Related	Informational	Mitigated	Xi_Zi
UNW-4	Check the return value of ERC20 token operations	Logical	Low	Fixed	yekong

UNW-5	Fee-On-Transfer tokens not supported in LiquidityPool on mint	Logical	Informational	Fixed	helookslike me
UNW-6	LiquidityPool can be broken by first depositor	Logical	Medium	Fixed	0xxm
UNW-7	LiquidityPool cannot be unpause	Logical	Low	Fixed	0xxm
UNW-8	Runtime deadline calculation allow pending transcations to be maliciously executed	Logical	Low	Acknowledged	0xxm
UNW-9	UniWhale - Token compatibility causes program errors in `LiquidityPool` contract	Logical	Informational	Acknowledged	Xi_Zi
UNW-10	`AbstractRegistry::_updateImbalanceNcePerPriceId` computation error when called twice in one block	Logical	Medium	Fixed	alansh
UNW-11	`AbstractRegistry::setFundingFeePerPriceId` computation error when called twice in one block	Logical	Low	Fixed	alansh
UNW-12	`AbstractRegistry` Set fee limit	Privilege Related	Low	Acknowledged	helookslike me
UNW-13	`LiquidityPool::mint` Use `safeTransferFrom`	Logical	Informational	Acknowledged	helookslike me
UNW-14	`RegistryCore::updateOpenOrder` Lack of salt validation	Logical	Low	Fixed	Xi_Zi
UNW-15	`TradingCore::createTrade` lack of `notPaused` modifier	Logical	Low	Mitigated	Xi_Zi
UNW-16	frontrun risk in `LiquidityPool` contract `mint` function	Race Condition	Informational	Acknowledged	alansh
UNW-17	funding fee should not be applied to margin in `TradingCoreLib._closeTrade` function	Logical	Informational	Declined	alansh
UNW-18	updateLatestPrice might cause loss for user	Logical	Low	Acknowledged	0xxm

UNW-1:APPROVED_ROLE is able to remove margin from user's order in TradingCore

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Medium	• code/packages/contracts/core-v1/contracts/TradingCore.sol#L219-L222	Mitigated	0xxm

Code

```
219:    _require(
220:      t.user == msg.sender || hasRole(APPROVED_ROLE, msg.sender),
221:      Errors.USER_SENDER_MISMATCH
222:    );
```

Description

0xxm : APPROVED_ROLE is allowed to removeMargin on behalf of user, which puts users' order in risk of forced liquidation. Meanwhile, the emitted event still records this operation by user.

```
function removeMargin(
    bytes32 orderHash,
    bytes[] calldata priceData,
    uint256 margin
  ) external payable notPaused nonReentrant {
    IRegistry.Trade memory t = registry.openTradeByOrderHash(orderHash);

    _require(
      t.user == msg.sender || hasRole(APPROVED_ROLE, msg.sender),
      Errors.USER_SENDER_MISMATCH
    );
    ...
    registry.updateOpenOrder(orderHash, trade);
    marginPool.transferBase(msg.sender, margin);
    emit UpdateOpenOrderEvent(t.user, orderHash, trade, false, margin);
}
```

Recommendation

0xxm : It is recommended to remediate this over-centralization issue by removing APPROVED_ROLE's privilege in `removeMargin`:

```
_require(
    t.user == msg.sender,
    Errors.USER_SENDER_MISMATCH
);
```

Client Response

APPROVED_ROLE is required so the extensions of TradingCore (e.g. TradingCoreWithRouter, not covered by the audit) can manage margin on behalf of users. It does somewhat increase the centralisation risk, but I don't think it deviates significantly from other centralisation risk we have. As a compromise, we can replace t.user with msg.sender in the event emission (so it is clear who called removeMargin)

UNW-2:Access control on view function is unnecessary

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/packages/contracts/core-v1/contracts/TradingCore.sol#L102	Acknowledged	0xxm

Code

```
102:  ) external view onlyRole(APPROVED_ROLE) returns (IRegistry.Trade memory) {
```

Description

0xxm : Function `createTrade` is a view function. There is no much sense to add access control to it, as contract code and storage on blockchain is public, and there is no way to prevent anyone from reading them.

```
function createTrade(
  OpenTradeInput memory openData,
  uint256 openPrice,
  uint256 slippage
) external view onlyRole(APPROVED_ROLE) returns (IRegistry.Trade memory) {
  return _createTrade(openData, openPrice, slippage);
}
```

Recommendation

0xxm : Remove `onlyRole` check in `createTrade` function.

Client Response

we agree that this restriction does not add to the security aspects of the contract. This was added from the product design perspective and for convenience purposes (to avoid accidental use), so we would like to keep the restriction.

UNW-3:Centralized risk

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Informational	• code/packages/contracts/core-v1/contracts/RegistryCore.sol#L7 • code/packages/contracts/core-v1/contracts/OracleAggregator.sol#L11 • code/packages/contracts/core-v1/contracts/iquidityPool.sol#L15 • code/packages/contracts/core-v1/contracts/TradingCore.sol#L18	Mitigated	Xi_Zi

Code

```
7:contract RegistryCore is AbstractRegistry {

11:contract OracleAggregator is AbstractOracleAggregator, PythParser {

15:contract LiquidityPool is

18:contract TradingCore is
```

Description

Xi_Zi : As there are privileged accounts of various roles in the contract, which play a key role in the contract, it is necessary to implement multi-signature protection for the accounts of various roles in the contract.

Recommendation

Xi_Zi : Multi-sign protection is required for the accounts of various roles of the contract.

Client Response

the centralisation risk is mitigated somewhat by the contract owner being a multisig/DAO contract wallet (e.g. Gnosis Safe).

UNW-4:Check the return value of ERC20 token operations

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L84 • code/packages/contracts/core-v1/contracts/iquidityPool.sol#L147	Fixed	yekong

Code

```
84:      ERC20(tokenIn).approveFixed(address(swapRouter), amountIn);

147:      baseToken.approveFixed(address(swapRouter), returnBalanceNet);
```

Description

yekong : The return value of the external call is not stored, and it is impossible to determine whether the authorization was successful

Recommendation

yekong : The return value of the external call is not stored, and it is impossible to determine whether the 'approve' was successful

Client Response

ERC20Fixed library now uses SafeERC20 to revert on unexpected behaviour/result.

UNW-5:Fee-On-Transfer tokens not supported in LiquidityPool on mint

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L83-L84	Fixed	helookslikeme

Code

```
83:     ERC20(tokenIn).transferFromFixed(msg.sender, address(this), amountIn);
84:     ERC20(tokenIn).approveFixed(address(swapRouter), amountIn);
```

Description

helookslikeme : Fee-on-transfer tokens lead to problems in mint

Recommendation

helookslikeme : Check amount of tokens received or disallow fee tokens from being used in the vault.

Client Response

we assume by "fee-on-transfer" tokens, you mean deflationary tokens. We now check the balance after the transfer before calling approveFixed. We will also whitelist what can be swapped into the baseToken as an extra pre-caution.

UNW-6:LiquidityPool can be broken by first depositor

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L99-L101	Fixed	0xxm

Code

```
99:    uint256 returnBalance = baseBalance == 0
100:       ? amountNet
101:       : amountNet.mulDown(balance).divDown(baseBalance);
```

Description

0xxm : Initial value of LP token can be manipulate by the first depositor, so that users may not receive shares for their deposit of baseToken.

Consider the following POC:

```
function mint(
  uint256 amountIn,
  uint256 amountOutMinimum,
  address tokenIn,
  uint24 poolFee
) public notPaused nonReentrant {
  uint256 baseBalance = _getBaseBalance();
  uint256 balance =  (this).totalSupplyFixed();
  uint256 amountGross = amountIn;

  if (tokenIn == address(baseToken)) {
    baseToken.transferFromFixed(msg.sender, address(this), amountGross);
  } else {
    ...
  }
  uint256 fee = amountGross.mulDown(mintFee);
  uint256 amountNet = amountGross.sub(fee);
  accruedFee += fee;
  uint256 returnBalance = baseBalance == 0
    ? amountNet
    : amountNet.mulDown(balance).divDown(baseBalance);

  _mint(msg.sender, returnBalance);
}
```

For simplicity of fixedpoint calculation, let us assume all tokens' decimal is 18. An attacker can exploit using these steps:

- Add 1 wei base token to LiquidityPool. Since both `fee` (see another issue for why fee can be zero) and `baseBalance` is zero, the attacker will get 1 wei LP token (returnBalance == amountNet == amountGross == amountIn).
- Transfer large amount of baseToken directly to the pool, such as 1e9 baseToken. Since no new LP token is minted, 1 wei LP token worths 1e9 baseToken.
- Normal users add liquidity to pool will receive 0 LP token if they add less than 1e9 token because of rounding division.

Recommendation

0xxm : - Uniswap V2 solved this problem by sending the first 1000 LP tokens to the zero address. The same can be done in this case i.e. when baseBalance == 0, send the first min liquidity LP tokens to the zero address to enable share dilution.

- In mint(), ensure the number of LP tokens to be minted is non-zero:

```
    uint256 returnBalance = baseBalance == 0
      ? amountNet
      : amountNet.mulDown(balance).divDown(baseBalance);
    require(returnBalance != 0, "No LP minted");
    _mint(msg.sender, returnBalance);
```

Client Response

we now check returnBalance != 0

UNW-7:LiquidityPool cannot be unpause

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L207-L209	Fixed	0xxm

Code

```
207:  function pause() external virtual onlyOwner {
208:    _pause();
209:  }
```

Description

0xxm : LiquidityPool only implements `pause()` but not `unpause()` function, meaning it will be lock forever if paused.

Consider below POC contract

```
function pause() external virtual onlyOwner {
    _pause();
}
```

Recommendation

0xxm : Introduce an `unpause()` in LiquidityPool contract:

```
function unpause() external virtual onlyOwner {
    _unpause();
}
```

Client Response

Fixed

UNW-8:Runtime deadline calculation allow pending transcations to be maliciously executed

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/SwapRouter.sol#L65 • code/packages/contracts/core-v1/contracts/SwapRouter.sol#L106	Acknowledged	0xxm

Code

```
65:         deadline: block.timestamp,

106:         deadline: block.timestamp,
```

Description

0xxm : The SwapRouter contract set the uniswap deadline as `block.timestamp` instead of off-chain paramter given by user, which enables pending transcations to be maliciously executed at a later point.

```
IUniswapV3.ExactInputSingleParams memory params = IUniswapV3
      .ExactInputSingleParams({
        tokenIn: input.tokenIn,
        tokenOut: input.tokenOut,
        fee: input.poolFee,
        recipient: msg.sender,
        deadline: block.timestamp,
        amountIn: input.amountIn /
          (10 ** (18 - ERC20(input.tokenIn).decimals())),
        amountOutMinimum: input.amountOutMinimum /
          (10 ** (18 - ERC20(input.tokenOut).decimals())),
        sqrtPriceLimitX96: 0
      });
```

Uniswap provides their users with an option to limit the execution of their pending actions. However, setting it to a runtime calculated `block.timestamp` will allow the swap transation to be executed as a any time.

This issue can be maliciously exploited is through MEV: Alice wants to add liquidity to the pool using token A that is not a base token, which invokes an internal transcation to swap A token to baseToken in uniswap. This transcation will be pending in the mempool if current fees are too high. The price of token A has increased significantly during the pending, meaning Alice would receive a lot more base token when the swap is executed. But that also means that her `amountOutMinimum` value is outdated and would allow for significant slippage. A MEV bot detects the pending

transaction. Since the outdated `amountOutMinimum` now allows for high slippage, the bot sandwiches Alice, resulting in significant profit for the bot and significant loss for Alice.

Recommendation

0xxm : Introduce a deadline parameter from user, instead of `block.timestamp`.

Client Response

we will implement this as part of our book of work.

UNW-9:UniWhale - Token compatibility causes program errors in `LiquidityPool` contract

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L70-L104	Acknowledged	Xi_Zi

Code

```
70:  function mint(
71:    uint256 amountIn,
72:    uint256 amountOutMinimum,
73:    address tokenIn,
74:    uint24 poolFee
75:  ) public notPaused nonReentrant {
76:    uint256 baseBalance = _getBaseBalance();
77:    uint256 balance = ERC20PausableUpgradeable(this).totalSupplyFixed();
78:    uint256 amountGross = amountIn;
79:
80:    if (tokenIn == address(baseToken)) {
81:      baseToken.transferFromFixed(msg.sender, address(this), amountGross);
82:    } else {
83:      ERC20(tokenIn).transferFromFixed(msg.sender, address(this), amountIn);
84:      ERC20(tokenIn).approveFixed(address(swapRouter), amountIn);
85:
86:      amountGross = swapRouter.swapGivenIn(
87:        ISwapRouter.SwapGivenInInput(
88:          tokenIn,
89:          address(baseToken),
90:          amountIn,
91:          amountOutMinimum,
92:          poolFee
93:        )
94:      );
95:    }
96:    uint256 fee = amountGross.mulDown(mintFee);
97:    uint256 amountNet = amountGross.sub(fee);
98:    accruedFee += fee;
99:    uint256 returnBalance = baseBalance == 0
100:      ? amountNet
101:      : amountNet.mulDown(balance).divDown(baseBalance);
102:
103:    _mint(msg.sender, returnBalance);
104:  }
```

Description

Xi_Zi : The mint function passes the external token through the tokenIn parameter, but does not consider token compatibility. If there is a token transaction deflation or inflation mechanism, the amount of tokens actually received by

the contract may not be consistent with the input.amountIn. However, amountIn is also used when the contract authorizes unswapV3, which may cause an error in exchange.

```
function mint(
    uint256 amountIn,
    uint256 amountOutMinimum,
    address tokenIn,//@audit
    uint24 poolFee
  ) public notPaused nonReentrant {
    uint256 baseBalance = _getBaseBalance();
    uint256 balance = ERC20PausableUpgradeable(this).totalSupplyFixed();
    uint256 amountGross = amountIn;

    if (tokenIn == address(baseToken)) {
      baseToken.transferFromFixed(msg.sender, address(this), amountGross);
    } else {
      ERC20(tokenIn).transferFromFixed(msg.sender, address(this), amountIn);//@audit
      ERC20(tokenIn).approveFixed(address(swapRouter), amountIn);//@audit

      amountGross = swapRouter.swapGivenIn(
        ISwapRouter.SwapGivenInInput(
          tokenIn,
          address(baseToken),
          amountIn,
          amountOutMinimum,
          poolFee
        )
      );
    }
    uint256 fee = amountGross.mulDown(mintFee);
    uint256 amountNet = amountGross.sub(fee);
    accruedFee += fee;
    uint256 returnBalance = baseBalance == 0
      ? amountNet
      : amountNet.mulDown(balance).divDown(baseBalance);

    _mint(msg.sender, returnBalance);
  }
```

Recommendation

Xi_Zi : Advised to run the afaterbalance-beforebalance command to check the number of tokens,Think more about token compatibility issues.

Client Response

we will implement whitelist to address the token compatibility issue.

UNW-10: `AbstractRegistry::_updateImbalancePerPriceId` computation error when called twice in one block

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/packages/contracts/core-v1/contracts/interfaces/AbstractRegistry.sol#L353-L358	Fixed	alansh

Code

```
353:      imbalance.uptoLastUpdate =
354:        (imbalance.current *
355:          int256(currentBlock - lastUpdate) +
356:          imbalance.uptoLastUpdate *
357:          int256(lastUpdate - imbalance.initialUpdate)) /
358:        int256(currentBlock - imbalance.initialUpdate);
```

Description

alansh : If there're two consecutive calls to `_updateImbalancePerPriceId` in the same block(**highly probable** as it's triggered by users), the net affect should be the same as only the second call is invoked. Otherwise the funding fee may be higher than expected.

Recommendation

alansh : Consider below fix in the `AbstractRegistry._updateImbalancePerPriceId()` function

```
        uint256 passedBlock = currentBlock - 1 - lastUpdate;
        if (passedBlock > 0) {
          imbalance.uptoLastUpdate =
          (imbalance.current * passedBlock) +
            imbalance.uptoLastUpdate *
            int256(lastUpdate - imbalance.initialUpdate) /
          int256(currentBlock - 1 - imbalance.initialUpdate);
        }
```

Client Response

Fixed

UNW-11: `AbstractRegistry::setFundingFeePerPriceId` computation error when called twice in one block

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/interfaces/AbstractRegistry.sol#L288-L293	Fixed	alansh

Code

```
288:      fundingFee.uptoLastUpdate =
289:        (fundingFee.current *
290:          (currentBlock - lastUpdate) +
291:          fundingFee.uptoLastUpdate *
292:          (lastUpdate - fundingFee.initialUpdate)) /
293:        (currentBlock - fundingFee.initialUpdate);
```

Description

alansh : If there're two consecutive calls to `setFundingFeePerPriceId` in the same block, the net affect should be the same as only the second call is invoked. Otherwise the user may be charge more fees than expected.

Recommendation

alansh : Consider below fix in the `AbstractRegistry.setFundingFeePerPriceId()` function

```
        uint256 passedBlock = currentBlock - 1 - lastUpdate;
        if (passedBlock > 0) {
          fundingFee.uptoLastUpdate =
              (fundingFee.current * passedBlock +
                fundingFee.uptoLastUpdate *
                (lastUpdate - fundingFee.initialUpdate)) /
              (currentBlock -1 - fundingFee.initialUpdate);
        }
```

More strictly speaking, `setFundingFeePerPriceId` should not be called twice in the same block, otherwise users may be charged abnormally, so the alternative fix is:

```
        uint256 passedBlock = currentBlock - 1 - lastUpdate;
        require(passedBlock > 0);
          fundingFee.uptoLastUpdate =
              (fundingFee.current * passedBlock +
                fundingFee.uptoLastUpdate *
                (lastUpdate - fundingFee.initialUpdate)) /
              (currentBlock -1 - fundingFee.initialUpdate);
```

Client Response

Fixed

UNW-12: `AbstractRegistry` Set fee limit

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Low	• code/packages/contracts/core-v1/contracts/interfaces/AbstractRegistry.sol#L260-L264	Acknowledged	helookslikeme

Code

```
260:
261:  function setFee(uint256 _fee) external onlyOwner {
262:    fee = _fee;
263:    emit SetFeeEvent(fee);
264:  }
```

Description

helookslikeme : Owners can set a very high `_fee` value, result the user has to pay very high cost.

Recommendation

helookslikeme : Set a maximum handling fee cap in the `setFee` function.

Client Response

we will implement this as part of our book of work

UNW-13: `LiquidityPool::mint` Use `safeTransferFrom`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L83	Acknowledged	helookslikeme

Code

```
83:     ERC20(tokenIn).transferFromFixed(msg.sender, address(this), amountIn);
```

Description

helookslikeme : The transferFrom() method is used instead of safeTransferFrom(), presumably to save gas. I however argue that this isn't recommended because:
OpenZeppelin's documentation discourages the use of transferFrom(), use safeTransferFrom() whenever possible

Recommendation

helookslikeme : Use OpenZeppelin's SafeERC20 library to increase the compatibility of token operations.

Client Response

ERC20Fixed library now uses SafeERC20 to revert on unexpected behaviour/result.

UNW-14: `RegistryCore::updateOpenOrder` Lack of salt validation

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/RegistryCore.sol#L109-L140	Fixed	Xi_Zi

Code

```
109:  function updateOpenOrder(
110:    bytes32 orderHash,
111:    Trade memory trade
112:  ) external override onlyRole(APPROVED_ROLE) {
113:    Trade memory t = _openTradeByOrderHash[orderHash];
114:
115:    _require(t.user == trade.user, Errors.TRADER_OWNER_MISMATCH);
116:    _require(t.priceId == trade.priceId, Errors.PRICE_ID_MISMATCH);
117:    _require(t.isBuy == trade.isBuy, Errors.TRADE_DIRECTION_MISMATCH);
118:
119:    _openTradeByOrderHash[orderHash] = trade;
120:    totalMarginPerUser[trade.user] = totalMarginPerUser[trade.user]
121:      .sub(t.margin)
122:      .add(trade.margin);
123:    minCollateral -= t.margin.mulDown(t.maxPercentagePnL);
124:    minCollateral += trade.margin.mulDown(t.maxPercentagePnL);
125:
126:    if (t.isBuy) {
127:      totalLongPerPriceId[t.priceId] -= t.leverage.mulDown(t.margin);
128:      totalLongPerPriceId[trade.priceId] += trade.leverage.mulDown(
129:        trade.margin
130:      );
131:      _updateLongImbalancePerPriceId(trade.priceId);
132:      _updateShortImbalancePerPriceId(trade.priceId);
133:    } else {
134:      totalShortPerPriceId[t.priceId] -= t.leverage.mulDown(t.margin);
135:      totalShortPerPriceId[trade.priceId] += trade.leverage.mulDown(
136:        trade.margin
137:      );
138:      _updateLongImbalancePerPriceId(trade.priceId);
139:      _updateShortImbalancePerPriceId(trade.priceId);
140:    }
```

Description

Xi_Zi : In the contract RegistryCore, the openMarketOrder function records the salt for each open order, but does not verify that the order's salt is consistent when updateOpenOrder is updated. salt validation is required to ensure that the order is updated correctly.

```
function openMarketOrder(
    Trade memory trade
  )
    external
    override
    onlyRole(APPROVED_ROLE)
    onlyApprovedPriceId(trade.priceId)
    returns (bytes32)
  {
    salt++;
    trade.salt = salt;
    bytes32 orderHash = keccak256(abi.encode(trade));
    openTradesPerPriceIdCount[trade.user][trade.priceId]++;
    openTradesPerUserCount[trade.user]++;
    totalMarginPerUser[trade.user] = totalMarginPerUser[trade.user].add(
      trade.margin
    );
    openTrades[trade.user][trade.priceId][trade.salt] = orderHash;
    _openTradeByOrderHash[orderHash] = trade;

    if (trade.isBuy) {
      totalLongPerPriceId[trade.priceId] += trade.leverage.mulDown(
        trade.margin
      );
      _updateLongImbalancePerPriceId(trade.priceId);
      _updateShortImbalancePerPriceId(trade.priceId);
    } else {
      totalShortPerPriceId[trade.priceId] += trade.leverage.mulDown(
        trade.margin
      );
      _updateLongImbalancePerPriceId(trade.priceId);
      _updateShortImbalancePerPriceId(trade.priceId);
    }

    minCollateral += trade.margin.mulDown(trade.maxPercentagePnL);

    return orderHash;
  }

    ...

function updateOpenOrder(
    bytes32 orderHash,
```

```
  Trade memory trade
    ) external override onlyRole(APPROVED_ROLE) {
      Trade memory t = _openTradeByOrderHash[orderHash];

      _require(t.user == trade.user, Errors.TRADER_OWNER_MISMATCH);
      _require(t.priceId == trade.priceId, Errors.PRICE_ID_MISMATCH);
      _require(t.isBuy == trade.isBuy, Errors.TRADE_DIRECTION_MISMATCH);
      _openTradeByOrderHash[orderHash] = trade;
      totalMarginPerUser[trade.user] = totalMarginPerUser[trade.user]
        .sub(t.margin)
        .add(trade.margin);
      minCollateral -= t.margin.mulDown(t.maxPercentagePnL);
      minCollateral += trade.margin.mulDown(t.maxPercentagePnL);

      if (t.isBuy) {
        totalLongPerPriceId[t.priceId] -= t.leverage.mulDown(t.margin);
        totalLongPerPriceId[trade.priceId] += trade.leverage.mulDown(
          trade.margin
        );
        _updateLongImbalancePerPriceId(trade.priceId);
        _updateShortImbalancePerPriceId(trade.priceId);
      } else {
        totalShortPerPriceId[t.priceId] -= t.leverage.mulDown(t.margin);
        totalShortPerPriceId[trade.priceId] += trade.leverage.mulDown(
          trade.margin
        );
        _updateLongImbalancePerPriceId(trade.priceId);
        _updateShortImbalancePerPriceId(trade.priceId);
      }
    }
  }
```

Recommendation

Xi_Zi : Verify the salt of the order when updateOpenOrder

`_require(t.salt == trade.salt, Errors);`

Client Response

Fixed

UNW-15: `TradingCore::createTrade` lack of `notPaused` modifier

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/TradingCore.sol#L98-L104	Mitigated	Xi_Zi

Code

```
98:  function createTrade(
99:    OpenTradeInput memory openData,
100:    uint256 openPrice,
101:    uint256 slippage
102:  ) external view onlyRole(APPROVED_ROLE) returns (IRegistry.Trade memory) {
103:    return _createTrade(openData, openPrice, slippage);
104:  }
```

Description

Xi_Zi : Without the notPaused modifier, you can still create an order if the contract is suspended

```
function createTrade(
    OpenTradeInput memory openData,
    uint256 openPrice,
    uint256 slippage
  ) external view onlyRole(APPROVED_ROLE) returns (IRegistry.Trade memory) {
    return _createTrade(openData, openPrice, slippage);
  }
```

Recommendation

Xi_Zi : Suggestion: Add the "notPaused" modifier to the createTrade function.

Client Response

this is mitigated by allowing access only to approved roles

UNW-16:frontrun risk in `LiquidityPool` contract `mint` function

Category	Severity	Code Reference	Status	Contributor
Race Condition	Informational	• code/packages/contracts/core-v1/contracts/iquidityPool.sol#L38-L52	Acknowledged	alansh

Code

```
38:  function initialize(
39:    address _owner,
40:    string memory _name,
41:    string memory _symbol,
42:    ERC20 _baseToken,
43:    AbstractRegistry _registry,
44:    ISwapRouter _swapRouter
45:  ) public initializer {
46:    __ERC20_init(_name, _symbol);
47:    AbstractPool._initialize(_owner, _baseToken, _registry);
48:    swapRouter = _swapRouter;
49:    mintFee = 0;
50:    burnFee = 0;
51:    accruedFee = 0;
52:  }
```

Description

alansh : There's a gap between `initialize` and `setMintFee`, scientist has a chance to frontrun.

Recommendation

alansh : Add a `_mintFee` parameter to `LiquidityPool.initialize` to ensure the mint fee is set once the logic contract is in effect.

```
  function initialize(
    address _owner,
    string memory _name,
    string memory _symbol,
    ERC20 _baseToken,
    AbstractRegistry _registry,
    ISwapRouter _swapRouter,
    uint256 _mintFee
  ) public initializer {
    __ERC20_init(_name, _symbol);
    AbstractPool._initialize(_owner, _baseToken, _registry);
    swapRouter = _swapRouter;
    mintFee = 0;
    burnFee = _mintFee;
    accruedFee = 0;
  }
```

Client Response

we will implement this as part of our book of work.

UNW-17:funding fee should not be applied to margin in `TradingCoreLib._closeTrade` function

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/packages/contracts/core-v1/contracts/libs/TradingCoreib.sol#L52	Declined	alansh

Code

```
52:         averageImbalance.mulDown(closePosition).add(closeMargin)
```

Description

alansh : It's not fair for users to be charged funding fee for non-leveraged margin, should only be applied to position.

Recommendation

alansh : Consider below fix in the `TradingCoreLib._closeTrade()` function

```
      onCloseTrade.accumulatedFee =
        averageFundingFee.mulDown(
          averageImbalance.mulDown(closePosition)
        ) *
        int256(block.number - trade.executionBlock);
```

Client Response

this is a business decision, not a security concern.

UNW-18:updateLatestPrice might cause loss for user

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/TradingCore.sol#L142-L149 • code/packages/contracts/core-v1/contracts/TradingCore.sol#L168-L174 • code/packages/contracts/core-v1/contracts/TradingCore.sol#L192-L198 • code/packages/contracts/core-v1/contracts/TradingCore.sol#L225-L231 • code/packages/contracts/core-v1/contracts/TradingCore.sol#L255-L261	Acknowledged	0xxm

Code

```
142:     uint256 updateFee = oracleAggregator.getUpdateFee(priceData.length);
143:     IOracleProvider.PricePackage memory pricePackage = oracleAggregator
144:       .updateLatestPrice{value: updateFee}(
145:       openData.priceId,
146:       openData.isBuy,
147:       priceData,
148:       updateFee
149:     );

168:     IOracleProvider.PricePackage memory pricePackage = oracleAggregator
169:       .updateLatestPrice{value: updateFee}(
170:       trade.priceId,
171:       !trade.isBuy,
172:       priceData,
173:       updateFee
174:     );

192:     IOracleProvider.PricePackage memory pricePackage = oracleAggregator
193:       .updateLatestPrice{value: updateFee}(
194:       t.priceId,
195:       !t.isBuy,
196:       priceData,
197:       updateFee
198:     );

225:     IOracleProvider.PricePackage memory pricePackage = oracleAggregator
226:       .updateLatestPrice{value: updateFee}(
227:       t.priceId,
228:       !t.isBuy,
229:       priceData,
230:       updateFee
231:     );

255:     IOracleProvider.PricePackage memory pricePackage = oracleAggregator
256:       .updateLatestPrice{value: updateFee}(
257:       t.priceId,
258:       !t.isBuy,
259:       priceData,
260:       updateFee
261:     );
```

Description

0xxm : Several functions, such as `openMarketOrder`, in TradingCore contract require user to send ether as updateFee for oracleAggregator. The exact fee is calculated by calling `oracleAggregator.getUpdateFee`. A user is likely to send more fee to TradingCore to ensure updateFee is sufficient. This might cause problems in two ways:

- This contract won't return excessive fee, which cause loss for user.
- Other user can misappropriate the residue in this contrac by sending less or no ether to update price.

```
function openMarketOrder(
    OpenTradeInput calldata openData,
    bytes[] calldata priceData
  ) external payable notPaused nonReentrant {
    _require(
      openData.user == msg.sender || hasRole(APPROVED_ROLE, msg.sender),
      Errors.USER_SENDER_MISMATCH
    );

    uint256 updateFee = oracleAggregator.getUpdateFee(priceData.length);
    IOracleProvider.PricePackage memory pricePackage = oracleAggregator
      .updateLatestPrice{value: updateFee}(
      openData.priceId,
      openData.isBuy,
      priceData,
      updateFee
    );

    _openMarketOrder(openData, pricePackage.price);
}
```

Recommendation

0xxm : Check `msg.value` is sufficient to cover updateFee, and return excessive ether at the end of function. An example of `openMarketOrder` could be:

```
function openMarketOrder(
    OpenTradeInput calldata openData,
    bytes[] calldata priceData
  ) external payable notPaused nonReentrant {
     ...
    uint256 updateFee = oracleAggregator.getUpdateFee(priceData.length);
    require(msg.value >= updateFee);
    ...
    _openMarketOrder(openData, pricePackage.price);

    // return remaining ether back to sender if applicable
    assembly {
      if gt(selfbalance(), 0) {
          let success := call(gas(), caller(), selfbalance(), 0, 0, 0, 0)
          if iszero(success) {
              revert(0, 0)
          }
      }
    }
}
```

Client Response

we will implement this as part of our book of work.

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 Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.