SB SECURITY

LiquidLoot - Points OTC

Security Review

Sep 9, 2025

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

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

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Overview

Project	Liquid Loot - Points OTC
Repository	Private
Commit Hash	96eef5be959c69cf50fa1ca36aab7d4647c9e284
Resolution	26851845f7ff8b45a009b0583031b7909399f6e5
Timeline	September 5 - September 8, 2025

Scope

PointsMarket.sol

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low/Info Risk	6

5. Findings

5.1. Low severity

5.1.1. Add settle order with signature

Severity: Low Risk

Description: Optional settlement function can be added for the convenience of the both parties, which can make it possible for the buyer to finalize order with the off-chain signature of the seller.

Recommendation: Here's a simple implementation, that can be used for reference:

```
function settleFilledWithSignature(
    uint256 orderId,
    uint256 nonce,
    uint256 deadline,
    bytes calldata signature
) public nonReentrant {
    MORE CODE

    require(token.status == STATUS_TOKEN_SETTLE, "Not settling");
    require(
        token.token != address(0) && token.settleRate > 0,
        "Token unset"
    );
    require(block.timestamp > token.settleTime, "Settle not started");
    require(order.buyer == msg.sender, "Only buyer");
    require(order.status == STATUS_ORDER_OPEN, "Order not open");
    require(block.timestamp <= deadline, "Signature expired");
    require(nonce == $.nonces[order.seller], "Invalid nonce");

    // Verify signature
    bytes32 structHash = keccak256(
        abi.encode(
            SETTLE_FILLED_TYPEHASH,
            orderId,
            nonce,
            deadline
        )
    );
    bytes32 hash = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, structHash));

    require(
        SignatureChecker.isValidSignatureNow(order.seller, hash, signature),
        "Invalid signature"
    );

    // Calculate settlement amounts
    MORE CODE

    // Effects
    order.status = STATUS_ORDER_SETTLE_FILLED;
    $.nonces[order.seller]++; // Increment nonce to prevent replay

    // Execute transfers and emit event
    MORE CODE
}
```

Resolution: Acknowledged

5.1.2. forceCancelOrder with native, if 1 of the parties reject the payment, it's blocked

Severity: Low Risk

Description: If either one of the parties rejects the payment, neither one will receive, due to the success check.

```
if (offer.exToken == address(0)) {
    if (buyerRefundValue != 0 && order.buyer != address(0)) {
        (bool ok1, ) = order.buyer.call{value: buyerRefundValue}("");
        require(ok1, "Refund buyer failed");
    }
    if (sellerRefundValue != 0 && order.seller != address(0)) {
        (bool ok2, ) = order.seller.call{value: sellerRefundValue}("");
        require(ok2, "Refund seller failed");
    }
}
```

Recommendation: Remove the success checks, as it's beneficial for both to ensure they can receive refunds.

Resolution: Fixed - The success checks were removed and pull-over-push function was implemented.

5.1.3. rounding in value and collateral conversions

Severity: Low Risk

Description: In

`forceCancelOrder`

```
uint256 buyerRefundValue = (order.amount * offer.value) / offer.amount; //rounding error ex: 512341e6 *
53422e6 / 112532e6
uint256 sellerRefundValue = (order.amount * offer.collateral) / offer.amount; // collateral
```

`settleCancelled`

```
uint256 collateral = (order.amount * offer.collateral) / offer.amount;
uint256 value = (order.amount * offer.value) / offer.amount;
```

`settleFilled`

```
uint256 collateral = (order.amount * offer.collateral) / offer.amount;
uint256 value = (order.amount * offer.value) / offer.amount;
```

`fillOffer`

```
toTransfer = (offer.value * amount) / offer.amount;
toTransfer = (offer.collateral * amount) / offer.amount;
```

Both seller and buyer amounts can be rounded, resulting in funds accumulating in the market and losses for the users. This happens because there's not enough precision in the calculations.

Recommendation: Increase the precision of the calculations.

Resolution: Acknowledged

5.1.4. Sell offer can be created with 0 collateral when pledgeRate < 100%

Severity: Low Risk

Description: The seller can create an offer that bypasses the collateral requirement. This can happen when giving a value, which, when multiplied by `pledgeRate` (when set to <100%) gives a smaller number than `WEI6`.

```
function newOffer(
  uint8 offerType,
  bytes32 tokenId,
  uint256 amount,//points
  uint256 value,//token
  address exToken,
  bool fullMatch
) external nonReentrant whenNotPaused {
  PointsMarketStorage storage $ = _s();

  require(
      offerType == OFFER_BUY || offerType == OFFER_SELL,
      "Bad offerType"
  );
  require(
      $.tokens[tokenId].status == STATUS_TOKEN_ACTIVE,
      "Invalid token"
  );
  require(amount > 0 && value > 0, "Invalid amount/value");
  require(
      exToken != address(0) && $.acceptedTokens[exToken],
      "Invalid quote token"
  );

  uint256 collateral = (value * $.config.pledgeRate) / WEI6;
  uint256 toTransfer = (offerType == OFFER_BUY) ? value : collateral;

  IERC20(exToken).safeTransferFrom(msg.sender, address(this), toTransfer);
```

Recommendation: 2 options - 1) Increase the precision, 2) add a minimum value enforcement.

Resolution: Fixed

5.1.5. Remove pausing from cancelOffer

Severity: Low Risk

Description: Emergency cancel cannot be performed by offerer, because it has `whenNotPaused` modifier.

Recommendation: Remove the modifier and allow creators cancel even when project is paused.

Resolution: Fixed

5.1.6. Convert all offer/order/settlement state constants to enums

Severity: Informational Risk

Description: Convert all the constants, related to the states of the structs to enums, as it's more readable and can be packed by type.

Recommendation: Although constants use less gas, grouping them by type will make the code easier to read.

Resolution: Acknowledged