# Trust Security

Smart Contract Audit

Hypercerts Marketplace

13/12/23

# Executive summary

**FINDINGS**

2, Low

3, High
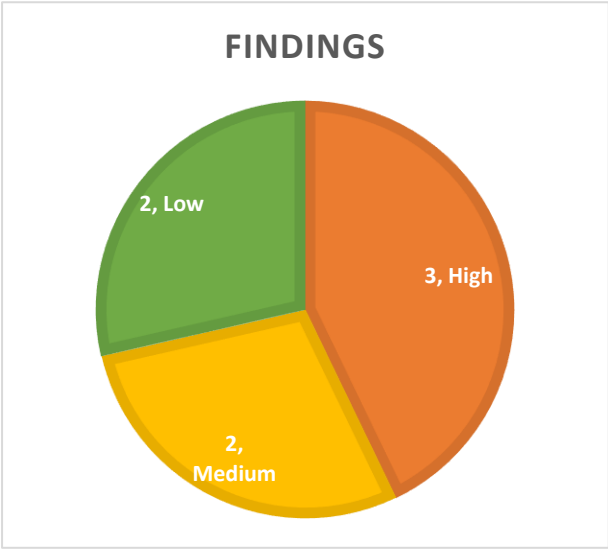
2, Medium

| Category | Marketplace |
|---|---|
| Audit type | Differential audit |
| Auditor | Trust |
| Time period | 04/12-13/12 |

Findings

| Severity | Total | Fixed | Acknowledged |
|---|---|---|---|
| High | 3 | - | - |
| Medium | 2 | - | - |
| Low | 2 | - | - |

Centralization score

Centralized                                                          Decentralized

Signature

# Document properties

## Versioning

| Version | Date | Description |
|---|---|---|
| 0.1 | 13/12/23 | Client report |

## Contact

**Trust**

trust@trust-security.xyz

# Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

## Scope

All changes between the LooksRare commit forked and the commit hash below are considered in scope.

## Repository details

- **Repository URL:** https://github.com/hypercerts-org/hypercerts
- **Commit hash:** 197968d71ff7782a4b3a9df8941666e8c466e7a9
- **Parent repository URL:** https://github.com/LooksRare/contracts-exchange-v2
- **Parent commit hash:** 7fca5650da653a2a0680e18e60164e639c18f830

## About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

## About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys assessing bounty contests in C4 as a Supreme Court judge.

## Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

## Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

# Qualitative analysis

| Metric | Rating | Comments |
| --- | --- | --- |
| Code complexity | **Good** | The code does not significantly increase the complexity of the forked codebase. |
| Documentation | **Good** | Project is mostly very well documented. |
| Best practices | **Good** | Project consistently adheres to industry standards. |
| Centralization risks | **Excellent** | Project does not introduce additional centralization risks. |

# Findings

## High severity findings

### TRST-H-1 A buyer can purchase more token units than the seller intended

- **Category:** Validation issues
- **Source:** StrategyHypercertFractionOffer.sol
- **Status:** Open

**Description**

In fraction offers, the maker provides the amount intended to be sold in **makerAsk.amounts[0]**. However, the strategy's processing logic only checks the collection's *unitsOf()* is above the taker's requested amount.

```
if (
    minUnitAmount > maxUnitAmount || unitAmount == 0 || unitAmount <
minUnitAmount || unitAmount > maxUnitAmount
        || pricePerUnit < makerAsk.price || makerAsk.price == 0
        || IHypercertToken(makerAsk.collection).unitsOf(itemIds[0]) <
unitAmount
) {
    revert OrderInvalid();
}
```

Suppose a user has 1000 tokens, and wishes to sell 499 of them, so they will always have majority vote in any governance based on the token units. An attacker can easily purchase 501 tokens and gain majority.

**Recommended mitigation**

Consider the maker's requested amount in the strategy validation.

**Team response**

TBD

### TRST-H-2 The fraction offer maker order is not invalidated correctly, leading to orders being replayed

- **Category:** Logical flaws
- **Source:** StrategyHypercertFractionOffer.sol
- **Status:** Open

**Description**

The marketplace supports three nonce types, including a per-order nonce which allows orders to be executed multiple times for partially filled orders. The **isNonceInvalidated** return value indicates the intention, below is the calculation in the fraction offer.

```
// If the amount to fill is equal to the amount of units in the
hypercert, we transfer the fraction.
// Otherwise, we do not invalidate the nonce because it is a partial
fill.
isNonceInvalidated =
IHypercertToken(makerAsk.collection).unitsOf(itemIds[0]) ==
unitAmount;
```

The nonce is only invalidated when the purchased amount (**unitAmount)** equals the total supply of the **itemID**. The logic is only sound under two assumptions:

- The user intended to sell the entire supply (an assumption also covered in H-1)
- The entire amount is purchased in a single order

For example, if a user intended to sell 50 units, and two buyers buy 1 and 49 units respectively, the order will remain valid. Suppose a user is interested in buying and holding tokens at a future point. An attacker can replay the previous maker order and purchase them at the previous price.

**Recommended mitigation**

Consider the remaining amount to be purchased at all time and invalidate when that amount is reduced to zero.

**Team response**

TBD

## TRST-H-3 When Hypercerts are traded in Collection or Dutch auction offers, one of the sides can provide a lower unit amount than expected

- **Category:** Logical flaws
- **Source:** StrategyCollectionOffer.sol,StrategyDutchAuction.sol
- **Status:** Open

**Description**

The marketplace supports trading of Hypercerts through the Collection and Dutch auction strategies. In this case they are viewed strictly as ERC1155s with an implicit **amount=1** enforced through the Hypercerts token contracts.

The strategies would function correctly for Hypercerts as long as the buyer and seller are in-sync on the amount of units transferred. However, it can be proven that can never be guaranteed to be the case, as Hypercert tokens can always be split by their owner.

Consider a Dutch auction for a Hypercert currently with 1000 units. A buyer may fill the order assuming they will receive 1000 units. However, the seller frontruns the order fill with a division of the units, so that 999 units are transported to a new **tokenID**, while 1 unit remains. The buyer therefore pays 1000x more than they expected per unit.

**Recommended mitigation**

The Hypercert implementation of pre-existing strategies must fork off and account for Hypercert units.

**Team response**

TBD

## Medium severity findings

### TRST-M-1 An attacker could grief buyer into getting a lower-valued item than intended

- **Category:** Validation flaws
- **Source:** StrategyCollectionOffer.sol
- **Status:** Open

**Description**

Hypercerts introduced the following strategy in StrategyCollectionOffer:

```
function executeCollectionStrategyWithTakerAskWithAllowlist(
    OrderStructs.Taker calldata takerAsk,
    OrderStructs.Maker calldata makerBid
)
```

The taker provides the **itemID** offered from the collection, the payment **recipient**, and a Merkle proof of the recipient's existence in the maker's Merkle tree. The issue is that anyone can execute the trade on recipient's behalf and pass a lower-valued **itemID**. The maker would end up overpaying as they assumed **recipient** would not offer a low-value item (they are whitelisted). The attacker does not stand to gain as they would need to provide the item whilst the payment goes to the whitelisted recipient. However due to the negative consequences for the maker and the possibly low cost of attack, the issue is still substantial.

**Recommended mitigation**

The taker parameters should additionally provide the recipient's signature over the **itemID** passed. Care should be taken to avoid any replay attacks.

**Team response**

TBD

### TRST-M-2 Fraction offers can be blocked from being fully fulfilled

- **Category:** Logical flaws
- **Source:** StrategyHypercertFractionOffer.sol
- **Status:** Open

**Description**

The Hypercerts marketplace supports selling of arbitrary-size fractions of a Hypercert until it is wholly consumed. The strategy checks the amount purchased is within the maker's defined minimum and maximum amount and that the price is not lower than the requested price.

```
//units, pricePerUnit
(uint256 unitAmount, uint256 pricePerUnit) =
abi.decode(takerBid.additionalParameters, (uint256, uint256));
//minUnitAmount, maxUnitAmount, root
(uint256 minUnitAmount, uint256 maxUnitAmount) =
abi.decode(makerAsk.additionalParameters, (uint256, uint256));
// A collection order can only be executable for 1 itemId but
quantity to fill can vary
if (
    minUnitAmount > maxUnitAmount || unitAmount == 0 || unitAmount <
minUnitAmount || unitAmount > maxUnitAmount
        || pricePerUnit < makerAsk.price || makerAsk.price == 0
        || IHypercertToken(makerAsk.collection).unitsOf(itemIds[0]) <
unitAmount
) {
    revert OrderInvalid();
}
```

Suppose that a holder wishes to sell 1000 units, at batches of size 100-200. After several sales, the remaining quantity is 99 units. This could occur naturally or base an attacker carefully picking a purchase amount. At this point, the holder cannot sell the remaining units under that order.

**Recommended mitigation**

Consider allows amounts less than **minUnitAmount** if they are the last portion of the order.

**Team response**

TBD

## Low severity findings

### TRST-L-1 The strategy validation function for fraction sales could revert

- **Category:** Validation issues
- **Source:** StrategyHypercertFractionOffer.sol
- **Status:** Open

**Description**

The *isMakerOrderValid()* function is implemented for every strategy and should return a **true** or **false** response. The following check in the fraction sale strategy isn't safe.

```
if (
    makerAsk.amounts.length != 1 || makerAsk.amounts[0] == 0
        ||
IHypercertToken(makerAsk.collection).unitsOf(makerAsk.itemIds[0]) <
makerAsk.amounts[0]
        || makerAsk.itemIds.length != 1 || minUnitAmount >
maxUnitAmount || makerAsk.price == 0
```

```
        || maxUnitAmount == 0
) {
    return (isValid, OrderInvalid.selector);
}
```

The key detail is that the **itemIds** array is accessed at index 0 before checking the array length. Therefore, on an empty array, the function would revert instead of returning **false**.

**Recommended mitigation**

Reverse the order of the validation checks.

**Team response**

TBD

## TRST-L-2 Hypercert orders with invalid amount will pass validations

- **Category:** Validation issues
- **Source:** BaseStrategy.sol
- **Status:** Open

**Description**

All strategies inherit from BaseStrategy and in their *isValidMakerOrder()* function, call the parent *validateAmountNoRevert()*:

```
/**
 * @dev This is equivalent to
 *      if (amount == 0 || (amount != 1 && collectionType == 0)) {
 *          return (0, OrderInvalid.selector);
 *      }
 * @dev OrderInvalid_error_selector is a left-padded 4 bytes. If the
error selector is returned
 *      instead of reverting, the error selector needs to be right-
padded by
 *      28 bytes. Therefore it needs to be left shifted by 28 x 8 =
224 bits.
 */
function _validateAmountNoRevert(uint256 amount, CollectionType
collectionType) internal pure {
    assembly {
        if or(iszero(amount), and(xor(amount, 1),
iszero(collectionType))) {
            mstore(0x00, 0x00)
            mstore(0x20, shl(224, OrderInvalid_error_selector))
            return(0, 0x40)
        }
    }
}
```

The function validates the order **amount**, and in case the collection is an NFT, it must be 1. However, since Hypercerts always have the ERC1155 **amount=1**, the validation should also perform that check for CollectionType.Hypercert.

**Recommended mitigation**

Consider improving validation of Hypercerts as suggested above.

**Team response**

TBD

## Additional recommendations

## TRST-R-1 Improve validation of orders in fraction offers

In the Hypercert fraction offer, *isMakerOrderValid()* checks the additional parameters byte array size is correct when using the allow list strategy.

```
if (
    functionSelector
        ==
StrategyHypercertFractionOffer.executeHypercertFractionStrategyWithTa
kerBidWithAllowlist.selector
        && makerAsk.additionalParameters.length != 96
) {
    return (isValid, OrderInvalid.selector);
}
isValid = true;
```

It is advised to have similar validation for the *executeHypercertFractionStrategyWithTakerBid()* strategy as well.

## Centralization risks

No centralization risks have been introduced on top of the LooksRare codebase.

## Systemic risks

### TRST-SR-1 Risks associated with Hypercerts

The marketplace supports working with Hypercert tokens, which involve inherent systemic risks. Most importantly they are upgradable, meaning any behavior that can be expected of Hypercerts can break if the owner opts for an upgrade, or is compromised.