SMART CONTRACT AUDIT REPORT

for

OTC Marketplace

Prepared By: Xiaomi Huang

PeckShield

June 26, 2025

Document Properties

Client	Drip
Title	Smart Contract Audit Report
Target	OTC Marketplace
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	June 26, 2025	Xuxian Jiang	Final Release
1.0-rc1	June 18, 2025	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the OTC Marketplace protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About OTC Marketplace

OTC Marketplace allows for trading of Hypurr NFTs, which are expected to be distributed to ~6000 addresses. These recipients are power users of the HyperLiquid DApp. Hypurr NFTs have not yet been released, and will be released at some unknown time in the future. However, even before the NFTs' release, this contract for audit aims to support an OTC market to handle pre-sales of these tokens. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of OTC Marketplace

Item	Description
Name	Drip
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	June 26, 2025

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/driptrade/otc-marketplace.git (c7d299c)

And this is the commit ID after all fixes for the issues found in the audit have been checked in.

- https://github.com/driptrade/otc-marketplace.git (fd97862, 859d8b9)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [6]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract

Table 1.3: The Full Audit Checklist

Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `OTC Marketplace` protocol. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	4	■■■■
Low	0	
Informational	0	
Total	4	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 4 medium-severity vulnerabilities.

Table 2.1: Key OTC Marketplace Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Improper NFT Token Validation in _buyItem()	Business Logic	Resolved
PVE-002	Medium	Incorrect Order Forfeiture Logic in OTCMarketplace	Business Logic	Resolved
PVE-003	Medium	Lack of Order Initialization Upon Item Sold/Bid Accepted	Business Logic	Resolved
PVE-004	Medium	Trust Issue of Admin Keys	Security Features	Mitigated

Besides the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Improper NFT Token Validation in _buyItem()

- ID: PVE-001
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `OTCMarketplace`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

`OTC Marketplace` allows sellers to list their to-be-received `Hypurr NFTs` for sale. In the process of analyzing the buyer's logic, we notice an issue that improperly validates the `Hypurr NFT` ownership.

To elaborate, we show below the code snippet of the related `_buyItem()` routine. As the name indicates, this routine is designed to buy a listed item. While this routine has performed extensive validation on the provided buy-item parameters, our analysis shows an extra validation is counter-productive. Specifically, it requires the buyer actually owns the listed `NFT` (line 918). As part of the design, the owner is the one who listed the `Hypurr NFT` for sale, not the buyer.

```
906         address _collateralToken = _listedItem.paymentTokenAddress;
907 
908         // Deplete listing quantity
909         if (_listedItem.quantity == _buyItemParams.quantity) {
910             delete listings[_buyItemParams.nftAddress][_buyItemParams.tokenId][
                    _buyItemParams.owner];
911         } else {
912             unchecked {
913                 listings[_buyItemParams.nftAddress][_buyItemParams.tokenId][
                        _buyItemParams.owner].quantity =
914                     _listedItem.quantity - _buyItemParams.quantity;
915             }
916         }
917 
918         _validateTokenIsTradeable(_buyItemParams.nftAddress, _buyItemParams.tokenId,
                _buyItemParams.quantity);
```

```
        _enterIntoEscrow(
            _storedPricePerItem,
            _buyItemParams.quantity,
            _msgSender(),
            _buyItemParams.owner,
            _buyItemParams.paymentToken,
            _collateralToken
        );
```

Listing 3.1: `OTCMarketplace::_buyItem()`

Recommendation Revisit the above routine to remove the need of validating the listed `NFT`.

Status The issue has been fixed by this commit: `20d5a43`.

3.2 Incorrect Order Forfeiture Logic in OTCMarketplace

- ID: PVE-002
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `OTCMarketplace`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

In `OTC Marketplace`, if a bid has been accepted, or a listing has been purchased, the funds are locked up in escrow until the `Hypurr NFTs` are distributed. Buyer or seller will not be able to back out of the trade. However, if a seller does not fulfill the order within a defined period after the `Hypurr NFTs` have been distributed, the seller forfeits the collateral to the buyer. While examining the forfeiture logic, we notice an issue that incorrectly validates the forfeiture requirement.

To elaborate, we show below the implementation from the related `forfeitOrders()` routine. By design, it can only forfeit an order if the marketplace contract is not paused and the fulfillment timestamp has elapsed. With that, we need to ensure the revert condition (line 577) to be `if (fulfillmentStartTimestamp == 0 || fulfillmentStartTimestamp + fulfillmentDuration >= block.timestamp)`, not current `if (fulfillment StartTimestamp == 0 || fulfillmentStartTimestamp + fulfillmentDuration < block.timestamp)`.

```
575    function forfeitOrders(OrderParams[] calldata _orderParams) external nonReentrant
           whenNotPaused {
576        // can only forfeit order if contract is not paused and and the timestamp has
               elapsed
577        if (fulfillmentStartTimestamp == 0  fulfillmentStartTimestamp +
               fulfillmentDuration < block.timestamp) {
```

```
            revert MarketplaceForfeitOrdersNotAllowed();
        }

        for (uint256 i = 0; i < _orderParams.length;) {
            _forfeitOrder(_orderParams[i]);

            unchecked { i += 1; }
        }
    }
```

Listing 3.2: `OTCMarketplace::forfeitOrders()`

Recommendation Revise the above routine to properly enforce the forfeiture requirement, i.e., it can only forfeit an order if the marketplace contract is not paused and the fulfillment timestamp has elapsed.

Status The issue has been fixed by this commit: `6104796`, `d0192ca`, and `335c0ca`.

3.3 Lack of Order Initialization Upon Item Sold/Bid Accepted

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `OTCMarketplace`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

As mentioned in Section 3.2, if a bid has been accepted, or a listing has been purchased, the funds are locked up in escrow until the `Hypurr NFTs` are distributed. Such deal or order is recorded in the internal `orders` array. In the process of analyzing the order bookkeeping logic, we notice an issue that does not properly record the order.

To elaborate, we show below the implementation of the related `_acceptBid()` routine. As the name indicates, this routine is invoked when a collection bid is accepted by the owner of a listing. We notice that the funds are locked up in escrow. However, it forgets to create a new order to record the deal (via the `_initOrder()` helper). In the meantime, once a new order is reordered, there is a need to update the associated `tokenMappings` data structure as well.

```
    function _acceptBid(
        AcceptBidParams memory _acceptBidParams,
        address _bidPaymentToken,
        uint128 _pricePerItem
    ) private {
```

```
        _validateTokenIsTradeable(_acceptBidParams.nftAddress, _acceptBidParams.tokenId,
             _acceptBidParams.quantity);

        _enterIntoEscrow(
            _pricePerItem,
            _acceptBidParams.quantity,
            _acceptBidParams.bidder,
            _msgSender(),
            _bidPaymentToken,
            _acceptBidParams.paymentToken
        );

        // Announce accepting bid
        emit BidAccepted(
            _msgSender(),
            _acceptBidParams.bidder,
            _acceptBidParams.nftAddress,
            _acceptBidParams.tokenId,
            _acceptBidParams.quantity,
            _acceptBidParams.pricePerItem,
            _bidPaymentToken,
            _acceptBidParams.bidType
        );
    }
```

Listing 3.3: `OTCMarketplace::_acceptBid()`

Recommendation Revisit the above routine to properly record the order and associated token mapping.

Status The issue has been fixed by this commit: `020a9e4`.

3.4 Trust Issue of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `InvestAITNew`
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

In the `OTC Marketplace` protocol, there is a special administrative account (with the `MARKETPLACE_ADMIN_ROLE` role). This administrative account plays a critical role in governing and regulating the system-wide operations (e.g., configure various settings, pause the protocol, manage payment tokens, and upgrade contracts). It also has the privilege to control or govern the flow of assets within the protocol contracts. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```
    function setCollectionApprovalStatus(
        address _nft,
        CollectionApprovalStatus _status
    ) external onlyRole(MARKETPLACE_ADMIN_ROLE) {
        if (_status == CollectionApprovalStatus.ERC_721_APPROVED) {
            if (!IERC165(_nft).supportsInterface(INTERFACE_ID_ERC721)) {
                revert MarketplaceCollectionDoesNotSupportInterface(_nft);
            }
        } else if (_status == CollectionApprovalStatus.ERC_1155_APPROVED) {
            if (!IERC165(_nft).supportsInterface(INTERFACE_ID_ERC1155)) {
                revert MarketplaceCollectionDoesNotSupportInterface(_nft);
            }
        }

        collectionApprovals[_nft] = _status;

        emit ApprovalStatusUpdated(_nft, _status, address(0));
    }

    ...
    function pause(bool paused) external onlyRole(MARKETPLACE_ADMIN_ROLE) {
        paused ? _pause() : _unpause();
    }

    ...
    function setFees(
        address _newFeeRecipient,
        uint256 _newBuyerFee,
        uint256 _newSellerFee
    ) public onlyRole(MARKETPLACE_ADMIN_ROLE) {
        if (_newFeeRecipient == address(0)) {
            revert MarketplaceAddressInvalid(_newFeeRecipient);
        }
        if (_newBuyerFee > BASIS_POINTS) {
            // buyer fee bps cannot exceed 10_000
            revert MarketplaceFeesTooHigh(_newBuyerFee, BASIS_POINTS);
        }
        if (_newSellerFee > BASIS_POINTS) {
            // seller fee bps cannot exceed 10_000
            revert MarketplaceFeesTooHigh(_newSellerFee, BASIS_POINTS);
        }

        feeRecipient = _newFeeRecipient;
        buyerFee = _newBuyerFee;
        sellerFee = _newSellerFee;

        emit UpdateFeeRecipient(_newFeeRecipient);
        emit UpdateFees(_newBuyerFee, _newSellerFee);
```

```
    }
```

Listing 3.4: Example Privileged Operations in `OTCMarketplace`

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to these privileged accounts may also be a counter-party risk to the contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Moreover, it should be noted that current contracts are to be deployed behind a proxy with a backend implementation. And naturally, there is a need to properly manage the admin privileges as they are capable of upgrading the entire protocol implementation.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changes to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been mitigated with the use of a multi-sig account to hold the admin account.

4 | Conclusion

In this audit, we have analyzed the design and implementation of the OTC Marketplace protocol, which allows for trading of Hypurr NFTs. These NFTs are expected to be distributed to ~6000 addresses, who are power users of the HyperLiquid DApp. Hypurr NFTs have not yet been released, and will be released at some unknown time in the future. However, even before the NFTs' release, this contract aims to support an OTC market to handle pre-sales of these tokens. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Moreover, we need to emphasize that Solidity-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[3] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[4] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[5] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[7] PeckShield. PeckShield Inc. https://www.peckshield.com.